

Atrenta Console User Guide

Version N-2017.12-SP2, June 2018



Copyright Notice and Proprietary Information

©2018 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

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 <http://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

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

Preface	17
About This Book	17
Contents of This Book	18
Typographical Conventions	19
Introducing Atrenta Console	21
Overview	21
Introducing Goals	21
Introducing Methodologies	22
Methodology Used by Atrenta Console	22
Before You Begin	23
Specifying Optional Environment Variables	23
Invoking Atrenta Console Graphical User Interface	24
Starting a New Session	24
Loading the Previous Session	24
Invoking Atrenta Console on a 64-bit Machine	25
GUI Details	26
Atrenta Console Flow	28
Design Setup Stage	28
Goal Setup and Run Stage	28
Analyze Results Stage	29
Files/Directories Created in Atrenta Console	30
Project File	30
Project Working Directory	30
Project Current Working Directory	31
File Generated in GUI	32
Files/Directories Generated by Default	32
Files Generated to Support Special Features	33
Using Atrenta Console Graphical User Interface	35
Overview	35
Stage 1: Setting up the Design (Design Setup)	36
Adding Design Files.....	36

Adding Files in GUI	36
Specifying Functionality Information of Gate Cells	42
Specifying a List of .sglib Files	44
Specifying Compressed Verilog Designs	44
Mapping File Extensions.....	45
Rearranging HDL Files	46
Performing Version Control.....	46
Editing Files	47
Setting Stop Files.....	50
Ignoring Files from SpyGlass Analysis.....	50
Waiving Messages by File.....	51
Viewing Include Files.....	51
Configuring Columns	51
Viewing and Changing Design Read Options	52
Using Verilog Constructs	54
Running Design Read	55
Running Design Read in GUI.....	55
Running Design Read in Batch	58
Checks Performed During the Design Read Process.....	58
Viewing Messages after Running Design Read	59
Identifying Common Syntax Errors and Issues	59
Tips for Debugging Syntax Errors.....	60
Viewing Reports.....	60
Viewing Source Files	62
Searching Instances.....	62
Stage 2: Selecting a Goal (Goal Setup & Run)	64
Selecting a Goal	65
Modifying a Goal.....	67
Running Custom Goals	72
Running Goals in Parallel.....	72
Viewing Directories Created After Goal Run	81
Setting up the Goal.....	93
Determining Parameter Precedence	93
Setting Parameters and Constraints for Selected Goal	93
Performing Sanity Checks for Parameters.....	101
Using the Dual Design Read (DDR) Flow.....	101
Incremental Mode Analysis.....	108
Setting Up the Goal in Batch Mode	109
The Methodology Configuration System	109
Running Prerequisite Goals.....	109
Working with Scenarios	111

Creating Scenarios.....	112
Modifying and/or Deleting Scenarios.....	114
Running Scenarios.....	114
Directory Structure Created After Running a Scenario.....	115
Stage 3: Analyzing a Design (Analyze Results)	116
Editing Source Files	117
Viewing Goal Summary	117
Comparing Results of Multiple SpyGlass Runs	118
Introducing the Incremental Mode Feature.....	118
Using the Incremental Mode Feature	119
Comparison Reported in Batch.....	120
Viewing Different Type of Results	120
Design Results	121
SpyGlass CDC Solution Results	122
SpyGlass Constraints Solution Results	122
SpyGlass TXV Solution Results	122
SpyGlass DFT Solution Results	124
Power Results	125
Viewing Results of Different Scenarios and Goals	126
Cross-probing from the Msg Tree Page.....	126

Working with Input Design and Libraries 129

Overview.....129

Working with Precompiled Libraries130

Advantages of Using Precompiling Libraries	130
Specifying Modes in Which Libraries Should be Compiled.....	130
Compiling HDL Files into a Library	131
Defining a Logical Library.....	131
Including HDL Files in the Logical Library	133
Generating a Precompiled Library	135
Automatically Compiling Gate Libraries	136
Using the GUI to Automatically Compile Libraries	137
Using the enable_gateslib_autocompile Option	138
Using the force_gateslib_autocompile Option.....	138
Using the AUTOENABLE_GATESLIB_AUTOCOMPILE Key.....	138
Specifying a Cache Directory.....	139
Conditions for Auto-Compilation of Gate Libraries.....	139
Built-in VHDL Libraries That Do Not Require Any Mapping.....	140
Precompiling Verilog Libraries.....	140
Naming and Mapping Verilog Libraries	141

Structure of Precompiled Verilog Libraries	142
Library Searching Mechanism	142
Working with Precompiled Verilog Libraries in Mixed Language Mode.....	143
Specifying Verilog Libraries by Using the 'uselib Statement	146
Compiling Libraries in Mixed-Language Designs.....	146
VHDL Library Design Units Instantiated in Verilog Modules.....	147
Verilog Modules Instantiated in VHDL Design Units	147
Searching Master Instance in Mixed-Language Mode.....	147
Debugging Issues in Gate Libraries.....	148
Specifying Precompiled Libraries for SpyGlass Analysis.....	149
Specifying Multiple Technology Libraries of the Same Name	150
Using Intermediate Logical Library Name Support in VHDL	151
Working with Compressed Gate Library Files.....	153
Working with Encrypted Compiled Libraries	154
Creating Encrypted Library Dump.....	154
Using Encrypted Library Dump.....	155
Viewing Built-In Messages for Precompiled Libraries	156
Impact of the addrules Option While Using Pre-compiled Dump	159
Impact of the ignorerules Option While Using Pre-compiled Dump	159
Mapping a File Extension with a Compilation Language	160
Inferring Language from File Extension During Compilation	161
Specifying Compilation Options in a Source File.....	165
Specifying Files in the Order of Their Dependencies	166
Compiling Verilog Files Containing SystemVerilog Keywords	167
Compiling the Set of Verilog and SystemVerilog Files Separately.....	167
Using File Extension Based Compilation Flow	168
Working with Encrypted Design Files	170
Introducing the Use Model for IP Encryption in SpyGlass	170
Encrypting IPs by Using the spyencrypt Utility	171
Arguments of the spyencrypt Utility.....	172
Encrypting IPs Spread Across a Hierarchical Directory Structure	175
Viewing Encryption Summary in a Report	176
Specifying Encrypted Files for SpyGlass Analysis	177
Specifying Encrypted Files through GUI.....	177
Specifying Encrypted Files through a Project File	180
Working with Mixed-Language Designs	182
Instantiating Verilog Modules in VHDL Architectures	182
Instantiating as Component Instance.....	182
Instantiating as Entity Instance	184
Instantiating VHDL Design Units In Verilog Modules.....	186
Examples of Instantiating VHDL Design Units in Verilog Modules	187

Mapping Data Types.....	189
Mapping between VHDL Generics and Verilog Parameters.....	189
Current Limitation with Mixed-language Designs in SpyGlass.....	190
Working with DesignWare® Modules.....	192
Prerequisites for Enabling DesignWare Flow.....	192
Specifying Path of DesignCompiler Installation.....	192
Enabling the DesignWare Flow.....	193
Reusing Netlist of DesignWare Modules during SpyGlass Analysis.....	194
List of DesignWare Modules Supported in SpyGlass.....	195
Using DesignWare Functions.....	198
Specifying Pragmas in HDL Code.....	199
Supported Pragmas for Verilog.....	199
Supported Pragmas for VHDL.....	199
Working with Black Boxes.....	200
Inferring Black Boxes.....	200
Understanding the Black Box Inference Feature.....	201
Using the Black Box Inference Feature.....	202
Checking the Inferred Information.....	202
Using the Corrected Inferred Information.....	204
Stopping Black Box Analysis.....	204
Handling Out of Memory Situations.....	205
Reporting Messages at Module Boundary.....	206
Identifying Modules.....	206
Enabling the Feature.....	207
Impact of the Feature.....	207
Controlling the RTL Synthesis Engine.....	208
Limiting Analysis of Memories.....	208
Preserving all instances and nets in a design.....	209
Interpreting Synthesis Pragmas.....	209
Interpreting Synthesis Pragmas.....	210
Managing the Design Hierarchy.....	212
Specifying a Top-level Design Unit.....	212
Advantage of Specifying a Top-Level Design Unit.....	212
Setting a Top-Level Design Unit.....	213
Multiple Top-Level Design Units.....	214
Language-Specific Behavior While Specifying a Top-Level Module.....	215
Stopping Design Units.....	216
Implications After Stopping Design Units.....	217
Checks Performed on Stopped Design Units.....	218
Using the Top and Stop Features Together.....	218

Ignoring Files and Design Units From SpyGlass Analysis.....	220
Difference between Ignored and Stopped Design Units.....	221
Ignoring Files Containing Design Units	221
Ignoring Individual Design Units	222
Analyzing Selective Design Hierarchy.....	225
Working with 'celldefine Modules.....	226
Performing Rule-Checking on 'celldefine Modules.....	226
Performing Hierarchical Rule-Checking in 'celldefine Modules	227

Working with Methodologies..... 229

Overview.....	229
Goal Files	230
Naming Convention of a Goal File.....	230
Details Present in a Goal File	231
Selection of Goal Files based on Language Mode	232
GuideWare Reference Methodology.....	234
Structure of the GuideWare Reference Methodology.....	234
Specifying an Active Methodology	238
Specifying a Current Methodology.....	242
Configuring a Methodology.....	244
Creating a Methodology	246
Modifying a Methodology	248
Creating and Modifying a Sub-methodology.....	249
Creating a Sub-Methodology	249
Modifying a Sub-Methodology	251
Creating Goals.....	251
Displaying the New Goals Dialog	253
Specifying Details in the New Goal Dialog.....	253
Importing Goals.....	255
Deleting Goals.....	256
Copying Goals	256
Modifying Goals.....	256
Modifying Goal Properties.....	257
Enabling/Disabling a Goal	259
Updating Rules of a Goal.....	259
Adding Rules in a Goal.....	261
Modifying Parameters of a Goal.....	263
Dragging and Dropping Sub-Methodologies and Goals.....	264
Creating Custom Methodologies	269

Customizing Goals	269
Including and Inheriting GuideWare Goals.....	269
Including/Inheriting Goals in a Goal File.....	270
Including/Inheriting Goals in the MCS Window.....	277
Viewing and Adding Options for an Included or Inherited Goal	278
Viewing Rules and Parameters of Included/Inherited Goals	279
Enabling/Disabling Rules of a Parent Goal	280
Selecting a Custom Methodology.....	281
Comparing Methodologies	284
Merging the Differences.....	286
Copying and Inheriting Methodologies	287
Copying a Methodology	287
Inheriting a Methodology	288
Specifying a Reference Environment Variable.....	289
Specifying an Additional Path	289
Migrating Custom Goals	291
Comparing Goals	291
Viewing the HTML Report for Comparison.....	296
Migrating Goals	297
Order File	298
Viewing Order of Goals Defined in an Order File	299
Format of an Order File	299
Map File	302
Working with SpyGlass Design Constraints	303
Overview.....	303
Specifying SGDC Files to SpyGlass	305
Creating an SGDC File	306
Adding Comments in an SGDC File.....	306
SGDC Convention for Packed Arrays.....	309
Specifying Multiple current_design Specifications for a Design Unit.....	309
Specifying Configuration Name with current_design Command.....	310
Specifying Multiple Values for a Constraint Argument.....	311
Handling Interdependencies between Different Arguments	311
Including an SGDC File in Another SGDC File.....	312
Specifying Signal Names.....	313
Specifying Signal Names based on Signal Types	313
Specifying Signal Names based on Design Hierarchy.....	313
Defining and Using Variables	315

Defining Variables	315
Using Variables.....	315
Handling Duplicate Constraint Specifications	318
Handling Nets Declared in a Sequential Block	319
Conditionally Specifying SGDC Constraints	320
Using the SG_OPERATING_MODE Variable.....	321
Example of Using the SG_OPERATING_MODE Variable	322
Processing of SGDC Files	327
Parsing SGDC Files.....	327
Performing Syntax Checking in SGDC Files	327
Processing SpyGlass Design and Waiver Pragmas	328
Recognizing Clocks.....	330
Converting SDC Attributes into SGDC Commands	331
Enabling the SDC-to-SGDC Translation Feature	331
Changing the Default Hierarchy Separator of the SDC2SGDC Constraints...	332
Specifying the Mode of Domain Inference.....	333
Inferring cdc_false_path for Clocks in Different Domains	334
Capturing Domain Inferring Results	335
Handling of Generated Clocks	336
Handling Mutually Exclusive Clocks.....	339
Handling Directional Clocks	339
Translating set_clock_sense command.....	340
Translating set_disable_timing command	340
Translating set_mode command.....	341
Saving the Generated SGDC Commands in a File	341
Specifying the Mode of an SDC File	341
Understanding Different Flows for Using This Feature.....	342
Generating SGDC Commands as a Part of Goal Run	342
Generating SGDC Commands as a Part of Design Read	342
Support for Virtual Clocks in sdc2sgdc Flow	343
Virtual to Real Clock Mapping	344
Limitations.....	344
Importing Block-Level SGDC Commands to Chip-Level	346
Creating a Migration File	346
Constraints Migrated From Block-Level to Chip-Level	347
Generated Hierarchical SGDC File(s).....	348
Validating the Generated Hierarchical SGDC File.....	349
Implementing Scoping in SGDC Commands.....	351
Wildcard Support at Top-Level	353

Conflict Resolution at Top-Level	353
Scoping When Design is at the Block-Level.....	354
Wildcard Support at Block-Level	355
Conflict Resolution at Block-Level.....	355
Handling SystemVerilog Objects in SGDC.....	357
Handling SystemVerilog Interface Port/Terminal.....	357
Handling SystemVerilog Interface Containing a Modport.....	357
Handling SV Structure or Union	358
Handling for-generate Constructs.....	359
Working with SpyGlass Messages	363
Overview.....	363
Working with Multiple Messages	365
Effects of Selected Messages in the Schematic.....	365
Selecting Static Auxiliary Messages	366
Selecting Non-Static Auxiliary Messages	366
Selecting Auxiliary Messages without Selecting a Main Message	367
Messages Affecting Multiple Source Lines/Files.....	367
Multiple Lines Affected in the Same Source File.....	368
Multiple Lines Affected in Different Source Files	368
Multiple Messages Selected	368
Limiting the Number of Messages Generated	369
Limiting the Number of Messages Reported for a Rule.....	369
Waiving Messages.....	371
Waiver File	372
Creating a Waiver File	372
Creating Goal-Based Waiver.....	372
Setting Default Waiver File.....	373
Handling Unsaved Changes in Waiver Files	374
Including a Waiver File in Another Waiver File.....	375
Effects of Waiving Messages.....	375
Auto-Migration of Waivers.....	376
Waiving Messages through GUI	377
Using the Waiver Editor Window	377
Using the Results Pane to Waive Messages	379
Waiving Messages through a Project File.....	381
Waiving Messages by Using the waive Constraint	381
Syntax of the waive Constraint	382
Argument Details of the waive Constraint.....	382

Details of the waive Constraint	386
Examples of Using the waive Constraint	389
Using Regular Expressions and Wildcard Characters.....	390
Support for Hierarchical Waivers	400
Waiving Messages by Using SpyGlass Pragmas	402
Waiving Rule Messages for a Block of Code.....	404
Waiving Rule Messages for a Single Line of Code.....	406
Ignoring the SpyGlass Waiver Pragmas	411
Waiving Messages in Waiver/SGDC Files.....	411
Existing Waiver Support in SpyGlass.....	412
Tagging Messages	413
Adding a Tag.....	413
Modifying a Tag	414
Deleting a Tag	415
Handling SpyGlass Built-In Messages.....	416
Handling Syntax Error Messages	416
Handling Language Warning Messages	416
Handling Synthesis Warning Messages	416
Handling Synthesis Error Messages	417
Handling Internal Messages.....	417
Working with Aggregated Reports	419
Overview.....	419
Searching for Input Files.....	420
Generating Aggregated Project Results.....	422
Project Summary Report	426
Generating the Project Summary Report	426
Generating the Report through GUI	426
Generating the Report through a Project File	427
Viewing the Project Summary Report.....	427
Viewing the HTML Report	427
Viewing CSV Reports.....	432
The DataSheet Report	433
Generating the DataSheet Report in GUI.....	435
Creating a Configuration File	437
Changing the Name of the Report	439
Adding a Logo in the Report Header	440
Tcl Format Support in the Configuration File.....	440
Generating the DataSheet Report in Batch	443

Generating the Datasheet Report through a Project File	444
Recommended Goals for Generating DataSheet Report	446
Details of the DataSheet Report.....	449
IO Definitions.....	451
Clock Trees	452
Reset Trees.....	453
Power	454
Power Clocks	455
Constraints	456
Testability.....	457
Design Statistics.....	458
Black Boxes	459
Timing	459
Congestion	460
The Dashboard Report	462
Licensing Requirements.....	464
Browser Compatibility	464
Generating Dashboard Report	464
Generating the DashBoard Report through Project File.....	465
Generating the DashBoard Report in Batch.....	465
Generating Dashboard Report in GUI.....	467
Creating a Configuration File	469
Tcl Format Support in the Configuration File	471
Creating the Success Criteria File.....	473
Viewing the DashBoard Report	492
Details of the DashBoard Report	493
SoC Dashboard	493
Module Dashboard	500
Customizing Report.....	504
Including Product-Specific Data in the Report	505
Customizing the Report Header	507
Managing Reports.....	509
Archiving and Managing Data Generated After Running Goals	509
Generating the HTML Goal Summary Page	511
Switching to the Old Dashboard Report.....	515
Goal Summary.....	516
Managing Datasheet and Dashboard Reports	518
Appendix.....	519
Supported HDL Directives	519

Re-using Simulation Scripts	521
Project File Details	523
Creating a Project File	523
Structure of a Project File	523
Data Import Section.....	524
Common Options Section	525
Goal Setup Section	529
Example of a Tcl-based Project File.....	534
Supported Library Cells	536
Combinational Cell Support	536
Sequential Cell Support	536
Precompiling Multiple Libraries in a Single SpyGlass Run.....	539
Features of Single Step Precompilation	540
Makefile Based Support in Step Precompilation	541
Combining Single-Step Precompilation and Top-level Run	542
Goals That Do Not Use Default Parameter Value	544
Sample Order File.....	555

Preface

About This Book

The Atrenta® Console User Guide describes how to use the Atrenta Console for rule-checking HDL designs.

Contents of This Book

The Atrenta Console User Guide has the following chapters:

Section	Description
<i>Introducing Atrenta Console</i>	About Atrenta Console
<i>Using Atrenta Console Graphical User Interface</i>	The Atrenta Console Basic Operating Principles
<i>Working with Input Design and Libraries</i>	Describes all aspects of reading a design in Atrenta Console.
<i>Working with Methodologies</i>	The Atrenta Console Methodology Configuration System (MCS)
<i>Working with SpyGlass Design Constraints</i>	SpyGlass Design Constraints Feature
<i>Working with SpyGlass Messages</i>	Features to control SpyGlass Messages
<i>Working with Aggregated Reports</i>	The Atrenta Console Reports
<i>Appendix</i>	Details of various other features in Atrenta Console

Typographical Conventions

This document uses the following typographical conventions:

To indicate	Convention Used
Program code	OUT <= IN;
Object names	OUT
Variables representing objects names	<sig-name>
Message	Active low signal name '<sig-name>' must end with _X.
Message location	OUT <= IN;
Reworked example with message removed	OUT_X <= IN;
Important Information	NOTE: This rule...

The following table describes the syntax used in this document:

Syntax	Description
[] (Square brackets)	An optional entry
{ } (Curly braces)	An entry that can be specified once or multiple times
(Vertical bar)	A list of choices out of which you can choose one
. . . (Horizontal ellipsis)	Other options that you can specify

Introducing Atrenta Console

Overview

Atrenta® Console is used to solve various design issues in the early stages of the design development process.

To solve design issues early in the design development cycle, Atrenta Console provides you a pre-packaged set of goals and methodologies called GuideWare™.

Introducing Goals

A goal is a pre-packaged set of rules that detects specific types of design issues. For example, the `connectivity` goal contains rules that checks for basic connectivity issues in a design. Similarly, the `simulation` goal contains rules that checks for basic simulation issues in a design.

When you run a goal after specifying design files in Atrenta Console all rules of that goal are run. Once the goal run is complete, appropriate violation messages are reported to indicate design issues.

Details of goals are specified in *Goal Files* (.spq files).

Introducing 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.

You can load the required methodology or sub-methodology that contains the required goals. For details on loading a methodology, see [Specifying an Active Methodology](#).

Methodology Used by Atrenta Console

By default, Atrenta Console uses the [GuideWare Reference Methodology](#) for design analysis.

GuideWare reference methodology provides guidance to 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 used during various phases of SoC design development flow (RTL, IP, and chip integration design phases).

You can also configure GuideWare reference methodology to map to your specific design style and hand-off requirements. For details, see [Working with Methodologies](#).

Before You Begin

Before using Atrenta Console, set the `ATRENTA_LICENSE_FILE` variable to a license server or a copy of the license file on a client machine.

NOTE: *It is recommended to set the `ATRENTA_LICENSE_FILE` variable in the `port@server` format rather than file name format.*

A standard licensing software from Acresso, which must be set in order to run SpyGlass®, controls SpyGlass.

By default, SpyGlass creates license jobs for the license files/servers specified using both the `LM_LICENSE_FILE` and `ATRENTA_LICENSE_FILE` variables. However, you can 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
```

Specifying Optional Environment Variables

You can set the following optional environment variables to customize SpyGlass GUI operations:

Environment Variable	Indicates...	Default Value
<code>SPYGLASS_HOME</code>	SpyGlass Home directory	<code><your-inst-dir>/SPYGLASS_HOME</code>
<code>ATRENTA_LICENSE_FILE</code>	SpyGlass license server	<code><port-number>@<hostname></code>

Invoking Atrenta Console Graphical User Interface

You can either start a new session or load a previous session while invoking Atrenta Console GUI.

Starting a New Session

To start a new session, use any of the following commands:

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

Once you specify any of the above commands, Atrenta Console GUI opens with a default project, as shown in [Figure 2](#).

Before exiting the session, you should save the project. A project is saved in a project file (.prj).

The project file contains details of the current GUI session so that this session can be restarted later with all the saved data. For more details on a project file, see [Project File](#).

Loading the Previous Session

To load a previous session, load a project created in that session.

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

You can load a project in either of the following ways:

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

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

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

Analyzes a design with the specified settings after invoking SpyGlass GUI.

You can also use the `-run` command-line option when you want to work in

Invoking Atrenta Console Graphical User Interface

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.

Invoking Atrenta Console on a 64-bit Machine

If you open a project file that uses libraries that were precompiled on 32-bit machine while invoking Atrenta Console on a 64-bit machine, Atrenta Console displays a dialog indicating that 32-bit file is being loaded on a 64-bit machine or vice-versa.

This dialog also lists the names of libraries that are not compatible on the current version (32-bit or 64-bit), as shown in the following figure:

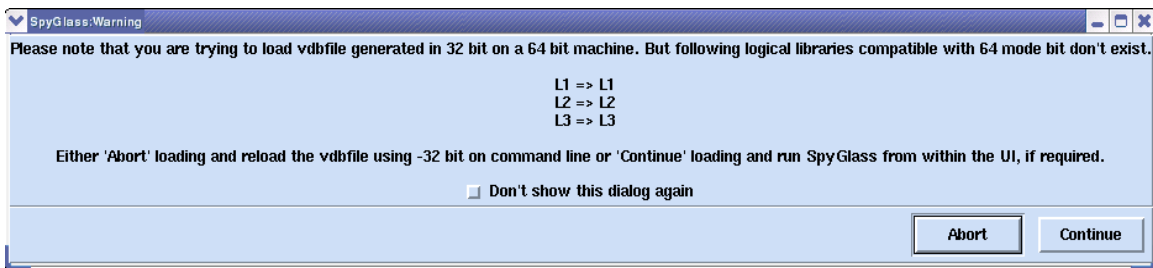


FIGURE 1. Warning Message

You can either abort the current session or continue the session by clicking the *Abort* or *Continue* buttons, respectively.

GUI Details

When you start a new session, the following page appears:

I

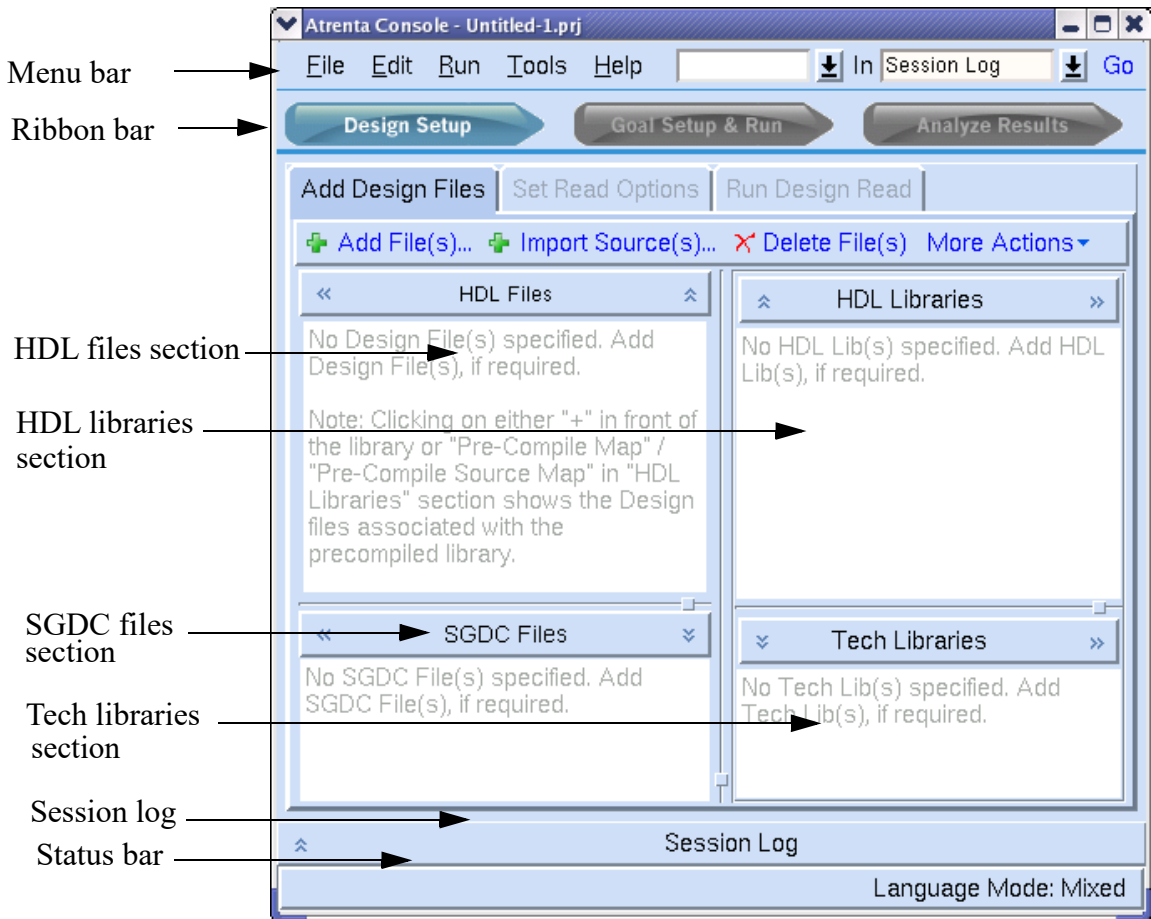


FIGURE 2. Atrenta Console GUI

Atrenta Console GUI contains the following elements:

- Menu bar that displays Atrenta Console menu options.

NOTE: *The menu bar is context sensitive and only displays the options that are relevant to the stage the design is in.*

- Ribbon bar that contains tab for the three stages (Design Setup, Goal Setup & Run, and Analyze Results) that enable you to analyze your design.

Once you complete a particular stage or specify the mandatory information in a particular stage, the tab for the successive stage gets enabled automatically. For example, once you specify design file(s) in the design setup stage, the *Goal Setup & Run* tab gets enabled automatically.

- *HDL Files* section that displays all the HDL files.
- *HDL Libraries* section that contains the HDL libraries and their corresponding RTL files and source file lists.
- *SGDC Files* section that displays all the SGDC files.
- *Tech Libraries* section that displays all the technology libraries.
- *Session Log* section that displays the GUI and runtime log.
- Status bar that displays the project state and summarizes the results.

Atrenta Console Flow

Atrenta Console flow is divided into the following three stages:

1. [Design Setup Stage](#)
2. [Goal Setup and Run Stage](#)
3. [Analyze Results Stage](#)

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, refer to the [Selecting a Goal](#) topic.

You can also specify the order in which goals should run. You can specify this order in an order file. For details, refer to the [Order File](#) topic.

- Provide additional design intent information, such as black boxes, clocks, and resets in your design. For details, refer to the [Central Design Setup](#) topic.

- Set the recommended parameters and required constraints for the selected goal. For details, refer to the [Setting up the Goal](#) topic.

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, refer to the [Working with SpyGlass Messages](#) chapter.
- Debug the reported issues by referring to schematics, reports, rule help, etc.
- Waive some violation messages if they do not indicate potential design issues. For details, refer to the [Waiving Messages](#) topic.

Add specific tags on some messages to debug such messages later.

Files/Directories Created in Atrenta Console

Atrenta Console 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 Atrenta Console.

Some files, however, are generated only when you use some special features. These files include [spyglass.db](#), [WORK](#), and [spyglass_cmdline_debug.log](#).

Project File

A project file (.prj) contains the following data about a particular Atrenta Console 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 Atrenta Console GUI, 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 an Atrenta Console batch session without using the Atrenta Console GUI.*

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`, Atrenta Console 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, Atrenta Console 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 Atrenta Console GUI, `spyglass.out` is generated.

This file is the screen-out file of Atrenta Console 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, Atrenta Console 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_spych**

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 Atrenta Console 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.

Atrenta Console generates a new file for each new run. However, if Atrenta Console 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.

Using Atrenta Console Graphical User Interface

Overview

This chapter discusses the following stages of Atrenta Console:

- *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 Atrenta Console.

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](#)
- [Mapping File Extensions](#)
- [Rearranging HDL Files](#)
- [Performing Version Control](#)
- [Editing Files](#)
- [Setting Stop Files](#)
- [Ignoring Files from SpyGlass Analysis](#)
- [Waiving Messages by File](#)
- [Viewing Include Files](#)
- [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 1 File Types Under the Add Design Files Tab

File	Description	Tcl Shell Usage
HDL files	<p>These files include Verilog (.v, .verilog, and .sv) files, VHDL (.vhd and .vhdl) files, Design Exchange Format (.def) files, and Library Exchange Format (.lef) files.</p> <p>NOTE: <i>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.</i></p>	<pre>read_file [-type <verilog vhdl hdl>] <filename></pre> <p>Note: <i>read_file <filename> is equivalent to read_file -type hdl <filename></i></p> <p>Example:</p> <pre>read_file -type verilog test.v</pre>
SGDC files	<p>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.</p>	<pre>read_file -type sgdc <filename></pre> <p>Example:</p> <pre>read_file -type sgdc constraints.sgdc</pre>
HDL libraries	<p>Contain precompiled VHDL or Verilog files.</p>	<pre>set_option lib <logical_name> <physical_path></pre> <p>Example:</p> <pre>set_option lib MyLib /a/b/mylib_path</pre>
Technology libraries	<p>Include Synopsys .lib, .sglib, .plib, and .gateslib files that are required for structured netlists using those library cells.</p>	<pre>read_file -type <gateslib sglib plib> <lib name></pre> <p>Example:</p> <pre>read_file -type sglib library.sglib</pre>
Source list files	<p>Include files with a .spp or .f extension. These files contain the design files, design options, or a combination of both.</p>	<pre>read_file -type sourcelist <file></pre> <p>Example:</p> <pre>read_file -type sourcelist sources.f</pre>

Steps for Adding Files

To add a file, perform the following steps:

1. Click the *Add File(s)* link under the *Add Design Files* tab.

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:

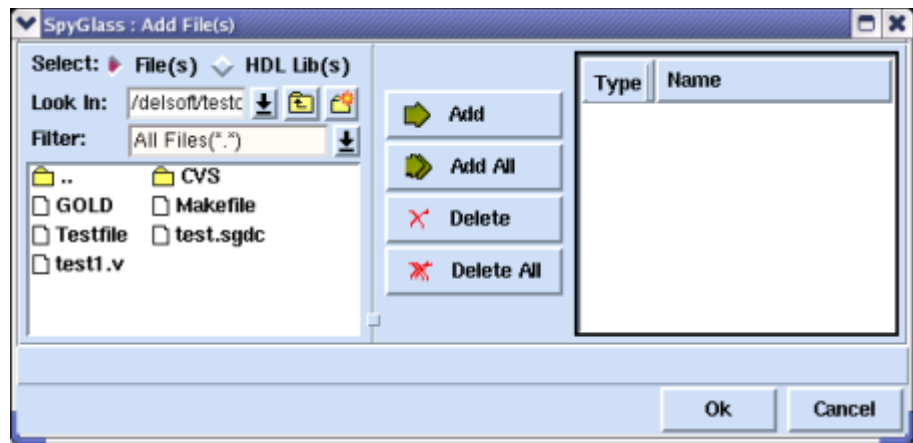


FIGURE 1. Add File(s) Dialog Box

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.

Stage 1: Setting up the Design (Design Setup)

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*, *SGDC Files*, *HDL Libraries*, and *Tech Libraries*, depending upon the type of the file selected.

The following figure shows various sections containing different files:

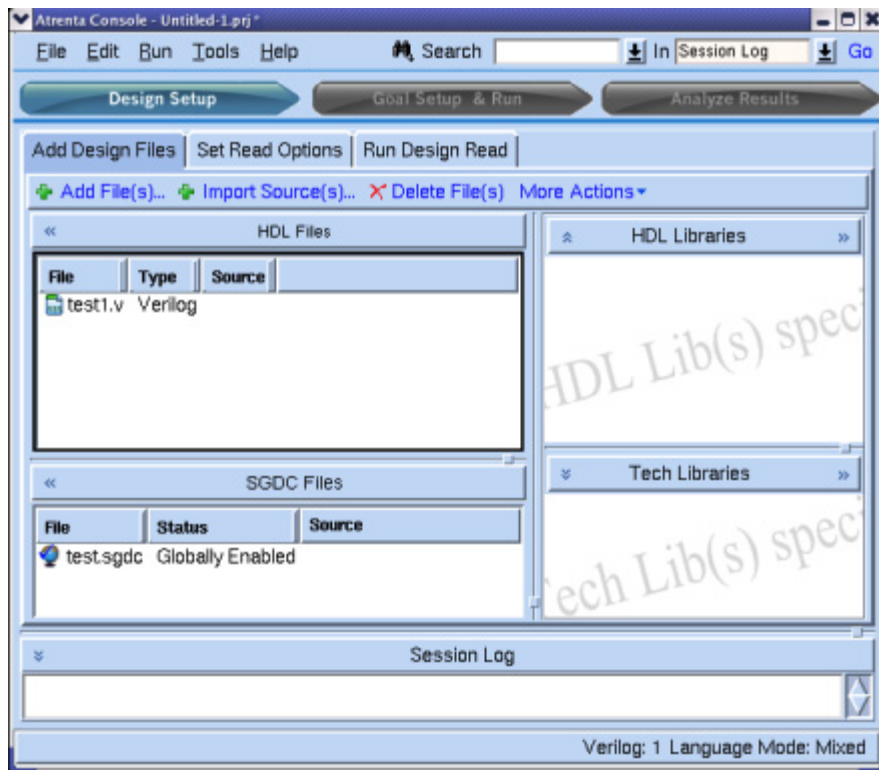


FIGURE 2. Atrenta Console

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 Atrenta Console.

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

Stage 1: Setting up the Design (Design Setup)

depending upon the file type. You cannot edit files that are grayed out.

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.

Atrenta Console 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 Atrenta Console 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 SGDC Files

By default, constraint files appearing under the *SGDC 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 SGDC file specific to that goal only. However, if you have changed the status of the SGDC file before selecting any goal, then it becomes a global SGDC 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.
- Select the file, and press the `<Delete>` key on the keyboard.

- 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 Atrenta Console. 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 Atrenta Console, 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>
```


Stage 1: Setting up the Design (Design Setup)

- Specify the name of the compressed file in a source file (.f file), and specify that source file in Atrenta Console by using the following project file command:

```
read_file -type sourcelist <source-file-name>
```

Mapping File Extensions

If you add an HDL file with an extension that is not recognized by SpyGlass, that file appears in red.

To enable SpyGlass to recognize such extensions, perform the following steps:

1. Click the *More Actions* link.
2. Select the *Edit File Extension Map* option from the drop-down menu.

The *File Extension Mapping* dialog appears as shown in the following figure:

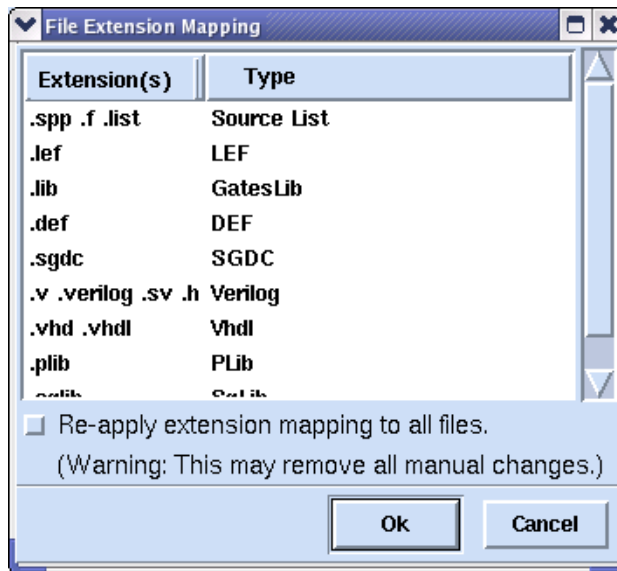


FIGURE 3. File Extension Mapping

3. In the above dialog, click the file type in the *Extension(s)* column.
The selected cell in the *Extension(s)* column becomes editable.
4. Now enter the required extension name in that cell in the *Extension(s)* column.

NOTE: *Specify extension names as a space-separated list.*

Rearranging HDL Files

To rearrange the order in which HDL files should appear in the *HDL File(s)* section, perform the following steps:

1. Select the desired file name.
2. Keeping the left mouse button pressed, drag the file up or down to the desired location.
3. Release the mouse button when you move the cursor to the desired location.

As you move the file, the cursor changes to indicate the drag/drop operation. As you move the file up and down, the target file appears in blue. The selected file always appears under the target file.

This feature is especially useful for arranging VHDL files where the order of processing is important.

In `sg_shell`, remove all files and add them back in the correct order.

NOTE: *You cannot rearrange the design files specified through source list files (.spp or .f).*

Performing Version Control

Performing version control provides the following benefits:

- Ensures that you are working on the latest HDL files
- Ensures that the changes you make in a file are not lost while modifying a design

To enable version control, configure a version control tool in SpyGlass by specifying appropriate details in the *Version Control* section of the *Preferences* dialog.

Once version control is enabled, the following options are displayed when

Stage 1: Setting up the Design (Design Setup)

you right-click on a file in the *File View* page.

- Check In

Use this option to place the file that you have edited in the version control tool.

- Check Out

Use this option to get a file from the version control tool.

- Get Latest

Use this option to get the latest version of a file from the version control tool.

NOTE: *The above-mentioned commands are related to CVS, which is the default version control tool used by SpyGlass. However, you can add or modify the existing commands based on your requirements. In addition, you can also integrate your own version control tool with SpyGlass.*

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.

Editing HDL Libraries

To edit an HDL library, perform the following steps:

1. Select the library in the *HDL Libraries* section.

Alternatively, you can select the *Edit Precompile Map* link from the *More Actions* drop-down list.

2. Specify the new alias of the file in the *Library Name* column.
3. Click (📁) to browse to the directory that contains the new library file.
4. Select the file
5. Click the *OK* button.

You must specify both the logical name and the path of the library. The logical name is the name of the library as you used it when creating a precompiled Verilog library or in your VHDL description. The physical library name is the complete path name of the library file.

If you change an entry in the *HDL Libraries* section, SpyGlass performs the following sanity checks:

■ **Filename existence check**

If you have not specified any RTL file for a particular precompile mapping, the corresponding library appears in red and the following message appears in a tool-tip:

No filename specified

■ **Empty file check**

SpyGlass checks the size of the RTL file being used. If the file size of all the specified RTL files is zero, the corresponding library appears in red and the following message appears in a tool-tip.

All files are of Zero size

■ **Read permission check**

If an RTL file being used is read-only, the corresponding library appears in red and the following message appears in a tool-tip.

Following files are not readable:

- *<file-name>*

Where, *<file-name>* is the name of the RTL file(s).

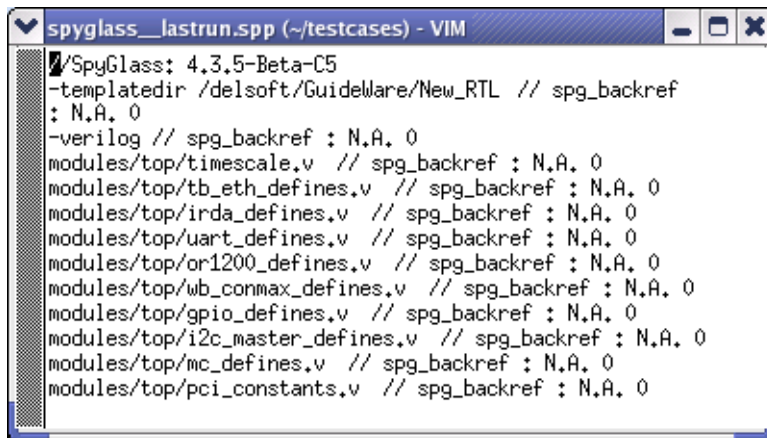
Modifying Source List Files

To modify a source list file, perform the following steps:

1. Right-click on the file, and select the *Edit Source List File* option from the shortcut menu.

The source list file opens in a text editor (set by you) as shown in the following figure:

Stage 1: Setting up the Design (Design Setup)



```

SpyGlass: 4.3.5-Beta-C5
-templatedir /delsoft/GuideWare/New_RTL // spg_backref
: N.A. 0
-verilog // spg_backref : N.A. 0
modules/top/timescale.v // spg_backref : N.A. 0
modules/top/tb_eth_defines.v // spg_backref : N.A. 0
modules/top/irda_defines.v // spg_backref : N.A. 0
modules/top/uart_defines.v // spg_backref : N.A. 0
modules/top/or1200_defines.v // spg_backref : N.A. 0
modules/top/wb_conmax_defines.v // spg_backref : N.A. 0
modules/top/gpio_defines.v // spg_backref : N.A. 0
modules/top/i2c_master_defines.v // spg_backref : N.A. 0
modules/top/mc_defines.v // spg_backref : N.A. 0
modules/top/pci_constants.v // spg_backref : N.A. 0

```

FIGURE 4. Source List File

2. Make the required changes in the file.
3. Save the source list file.

You can reload a source list file in Atrenta Console by right-clicking a source file and selecting the *Reload Source List File* shortcut menu option. Atrenta Console then re-reads the included source list files and updates the design settings based on the changes made.

Importing Options in a Project File


You cannot edit the options (design files and other options) mentioned in the source list files. You can, however, import these options in the project file.

To import the options in a project file, right-click on the source file and select the *Import From Source List Files* option from the shortcut menu. After performing this action:

- Reference of these files to the source list file is broken.
- Atrenta Console considers the file as a normal imported design file (Added using the *Import Sources...* option) and adds all its options to the project file.

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  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.

Stage 1: Setting up the Design (Design Setup)

Waiving Messages by File

To waive messages by file name, right-click on the file name and select the *Waive Messages by File Name* option from the shortcut menu. This displays the *Waivers* dialog in which the selected file appears in a separate row.

For details, see *Tools > Waiver Editor* section in the *Atrenta Console Reference Guide*.

Viewing Include Files

Some files displayed in the *HDL Files* section may contain include files. To identify files included in a file appearing in the *HDL Files* section, select the file name and click the *Show Include File(s)* option in the *Add Design Files* tab.

This displays the *Showing Include file(s)* dialog, as shown in the following figure:

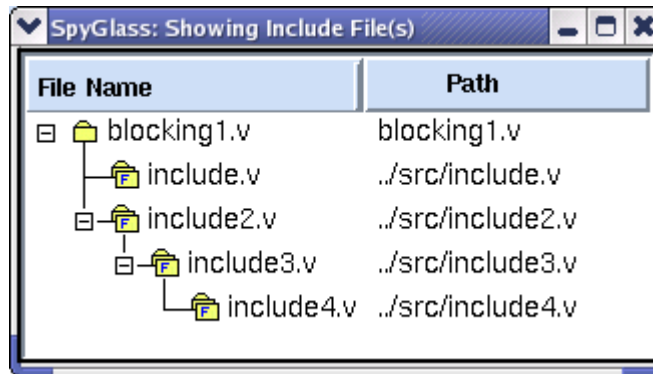


FIGURE 5. Include File(s)

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.

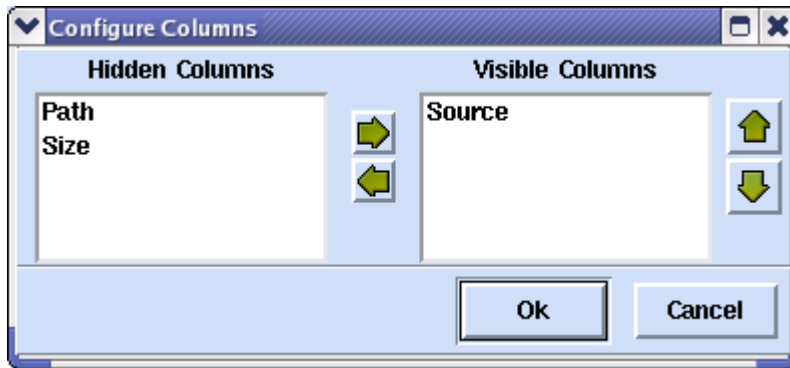






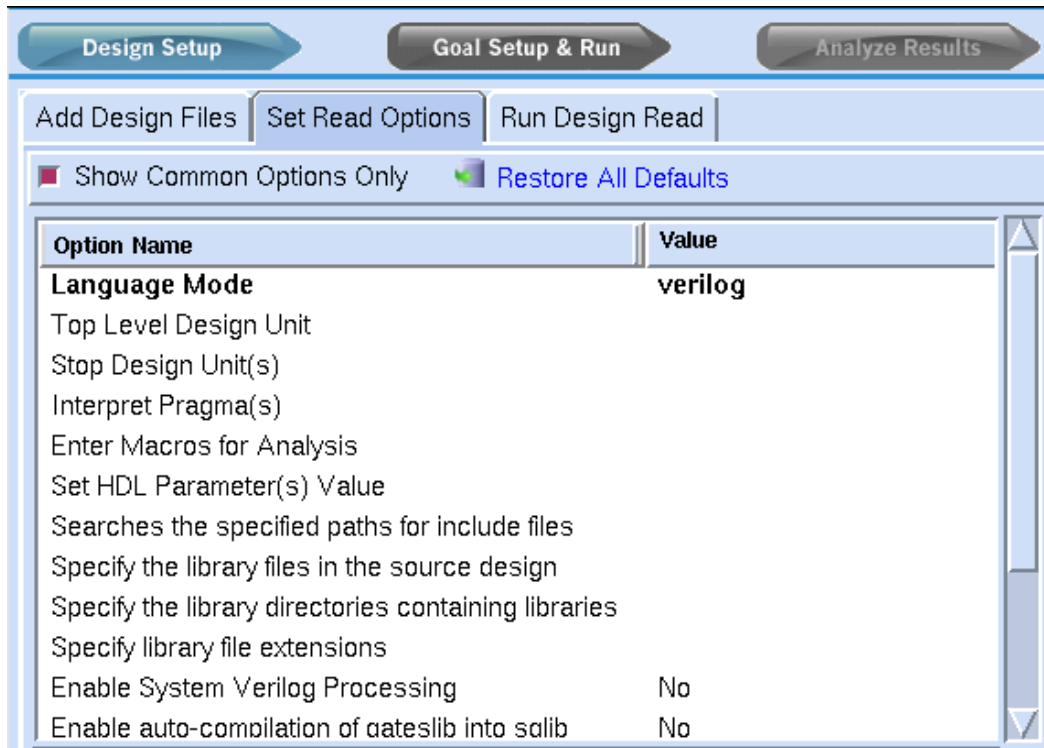
FIGURE 6. Configure Columns

2. Select the column name in the *Hidden Columns* or *Visible Columns* section.
3. Click  or  to show or hide the selected column.
4. You can also rearrange the column order in the *Visible Columns* section by clicking  or  buttons.

Viewing and Changing Design Read Options

Click the *Set Read 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:

Stage 1: Setting up the Design (Design Setup)

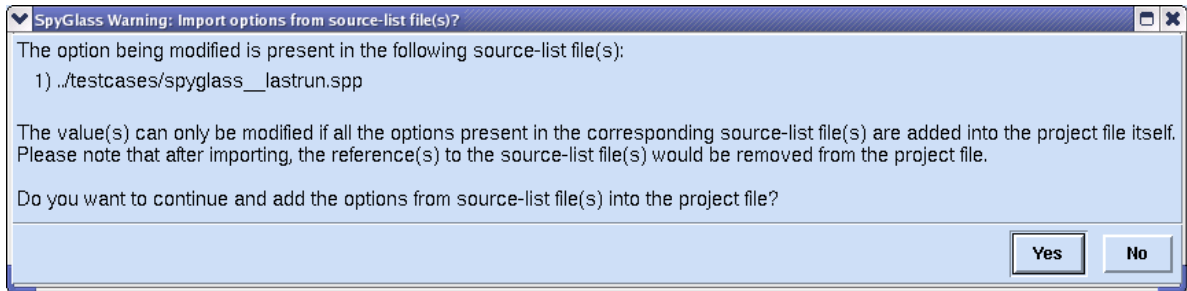
**FIGURE 7.** Design Read Options

By default, Atrenta Console displays only the commonly-used read options. To view all the options, deselect the *Show Common Options Only* check box.

When you select an option, the help related to that option appears in the *Option Help* section.

Refer to *Atrenta Console Reference Guide* for details on all of the design read options.

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.*

**FIGURE 8.** Warning Message

Using Verilog Constructs

By default, Atrenta Console assumes that you are using the Verilog 1364-2001 constructs.

Atrenta Console provides synthesis support for the following Verilog 2001 constructs:

TABLE 2 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 2 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 *Run Design Read* tab.

The following page appears:

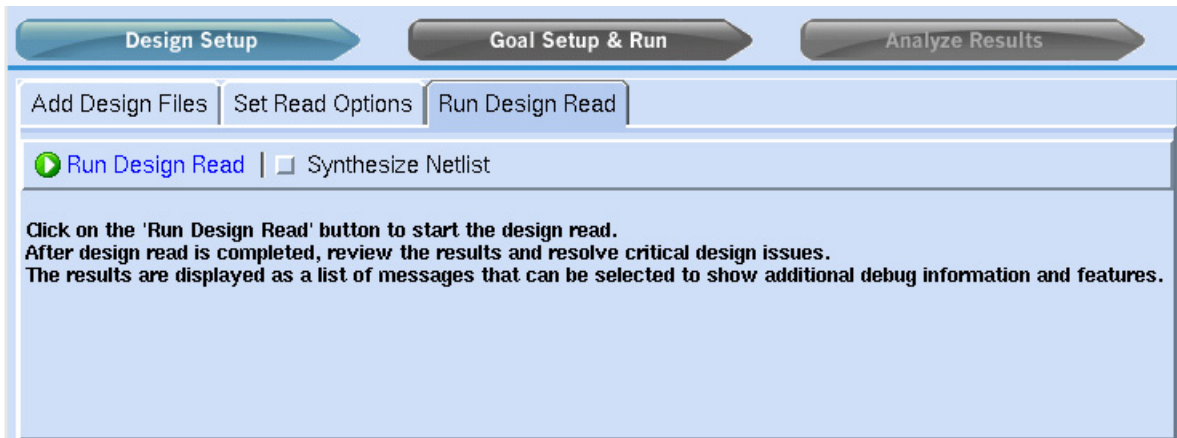


FIGURE 9. Run Design Read Tab

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.

NOTE: *If you do not save the project file before running the design read process, Atrenta Console prompts you to save the file.*

When the HDL analysis is complete, Atrenta Console displays the message information in the *Message Window* as shown in the following figure:

Stage 1: Setting up the Design (Design Setup)

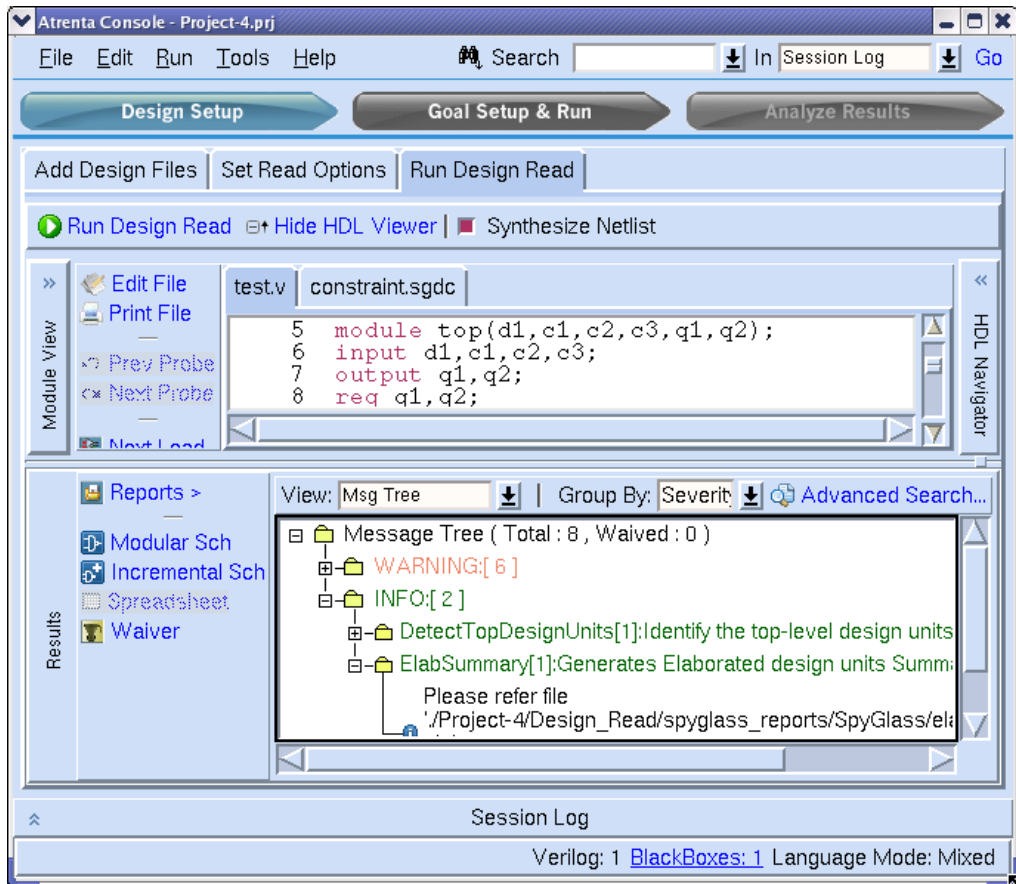


FIGURE 10. Run Design Read-Messages Window

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:

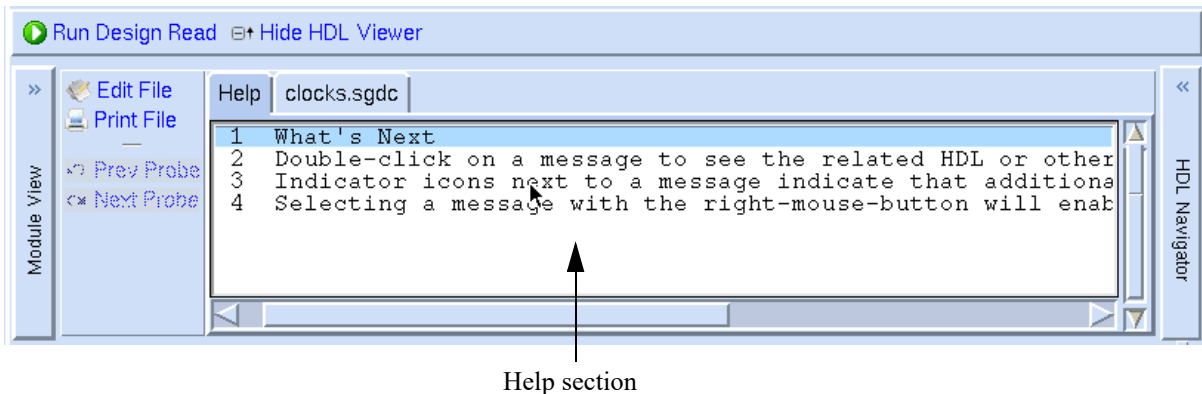


FIGURE 11. Run Design Read-Help Section

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

Atrenta Console 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 Atrenta Console 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*

Stage 1: Setting up the Design (Design Setup)

avoid problems during later analysis steps.

For a more complete example of a project file, refer to the [Project File Details](#) section.

Viewing Messages after Running Design Read

After the design read process, Atrenta Console 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

Atrenta Console reports syntax errors in the following cases:

- If a macro is referenced in a design before (or without) being declared, Atrenta Console reports the `STX_VE_533` syntax error.
- If you do not correctly specify a Verilog include directory, Atrenta Console 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, Atrenta Console 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, Atrenta Console 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.

Stage 1: Setting up the Design (Design Setup)

- 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 12. 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 or click the message on the *Message List* window of the *Msg Summary* page, the source file appears in the *source* section and the design unit instances appear on the *Instance View Page* and the *Module View Page*. 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*:

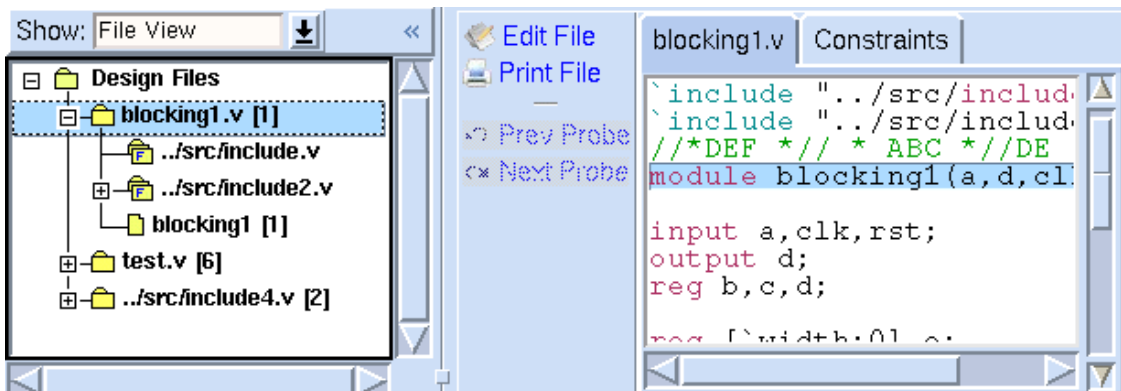


FIGURE 13. View Source Files

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:

Stage 1: Setting up the Design (Design Setup)

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 14. 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:

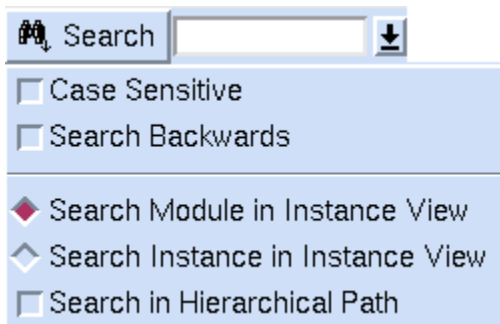


FIGURE 15. Search Options

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:

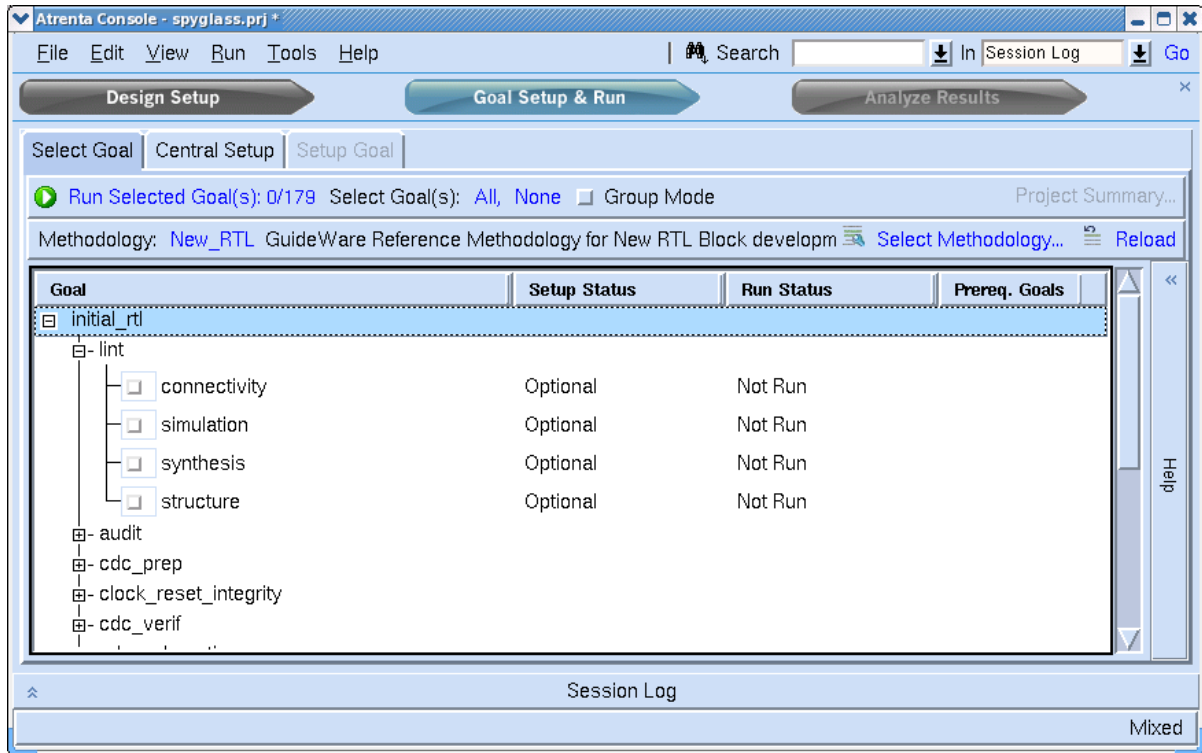


FIGURE 16. Goal Setup and Run

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)*.

Stage 2: Selecting a Goal (Goal Setup & Run)

This stage is divided into the following steps:

1. [Selecting a Goal](#)
2. [Central Design Setup](#)
3. [Setting up the 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:

Goal	Setup Status	Run Status	Prereq. Goals
<ul style="list-style-type: none"> [-] initial_rtl <ul style="list-style-type: none"> [-] lint [-] audit [-] cdc_prep [-] clock_reset_integrity [-] cdc_verif [-] cdc_exhaustive [-] constraint_generation [-] constraint [-] txv_verification [-] power 			

FIGURE 17. Goal Selection

NOTE: The default methodology is *GuideWare New_RTL*.

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 3

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 <i>Setup Optional</i> , you may or may not set the parameters for that goal. However, if the setup status of a goal is <i>Setup Recommended</i> , this means that the goal requires some additional steps. SpyGlass enables you to perform these steps through a setup wizard (see Setting up the Goal section for details).
Run Status	Displays whether the selected goal was run. When you click the <i>Run Selected Goal(s)</i> link, the status of the selected goal changes to <i>Running</i> . However, when the run is completed, then the run status changes to <i>Completed</i> . 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: <i>The run status also appears on the status bar.</i>
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).

The Atrenta Console 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

Stage 2: Selecting a Goal (Goal Setup & Run)

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, refer to the [design-read Synthesis Flavor](#) section of the *Atrenta Console Reference Guide*.

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 *Enable/Disable Rules* option from the shortcut menu. This displays the *Edit Rules* dialog containing all rules (enabled and disabled) for the selected goal.

The following figure shows the *Edit Rules* dialog for the *connectivity* goal:

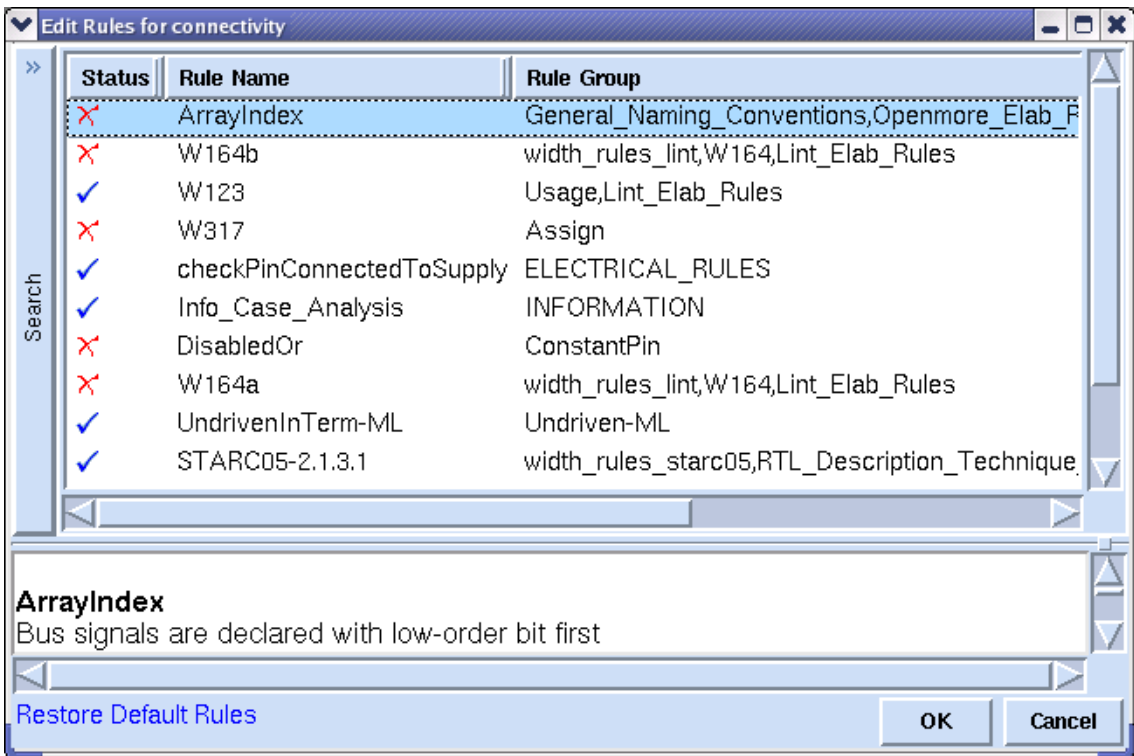


FIGURE 18. Edit Rules for Connectivity

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.

Stage 2: Selecting a Goal (Goal Setup & Run)

The following figure shows the *Edit Rules* dialog for the connectivity goal:

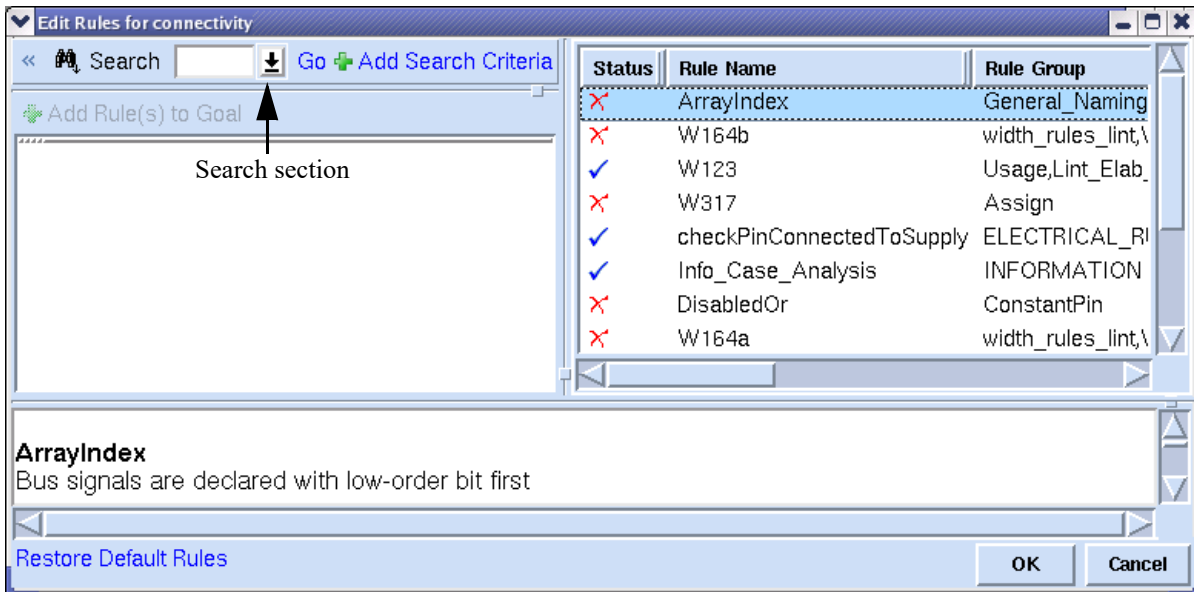


FIGURE 19. Edit Rules for Connectivity-Search

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:

Rule Name	Rule Group	Policy	Severity Label	Severity Class
W143	Array,Usage	lint	Warning	WARNING
W146	width_rules_lin	lint	Guideline	WARNING

Status	Rule Name	Rule Group
✗	ArrayIndex	General_Naming_Cr
✗	W164b	width_rules_lint,W16
✓	W123	Usage,Lint_Elab_Ru
✗	W317	Assign
✓	checkPinConnectedT	ELECTRICAL_RULE
✓	Info_Case_Analysis	INFORMATION
✗	DisabledOr	ConstantPin
✗	W164a	width_rules_lint,W16

FIGURE 20. 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.

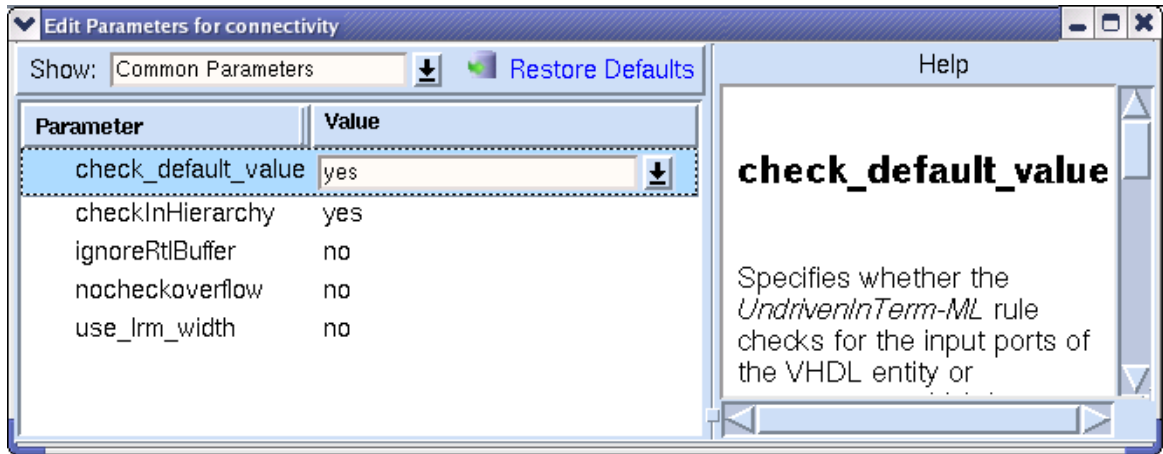
After performing the above steps, the specified rules appear in the selected goal.

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:

Stage 2: Selecting a Goal (Goal Setup & Run)

**FIGURE 21.** 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	Setup Status	Run Status	Prereq. Goals
<input checked="" type="checkbox"/> initial_rtl			
<input checked="" type="checkbox"/> detailed_rtl			
<input checked="" type="checkbox"/> rtl_handoff			
<input checked="" type="checkbox"/> ip_handoff			
<input type="checkbox"/> CUSTOM_GOAL_1	Setup Optional	Not Run Yet	

FIGURE 22. 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:

Stage 2: Selecting a Goal (Goal Setup & Run)

- 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, refer to *Atrenta Console Reference Guide*.

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:

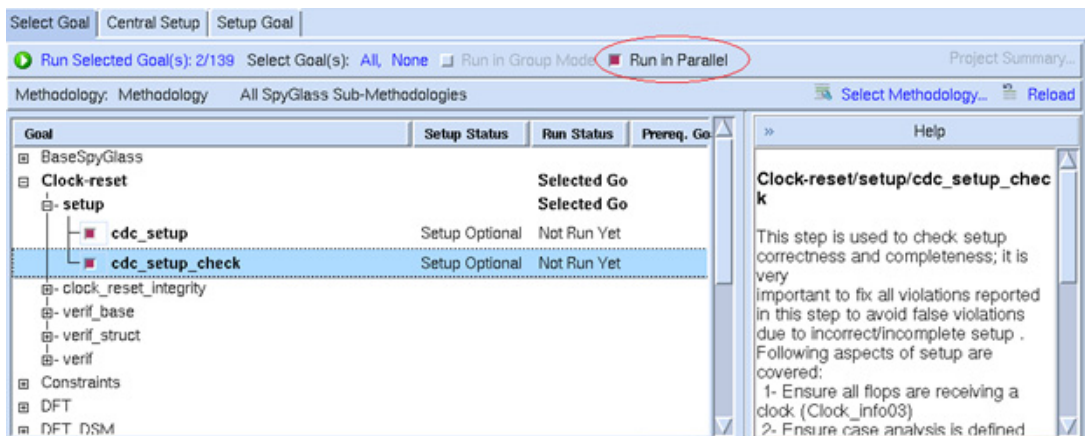


FIGURE 23. Run Selected Goal(s)

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`.

Stage 2: Selecting a Goal (Goal Setup & Run)

- 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`.
- Atrenta Console 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 is the 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 <code>bsub</code> .

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:

Stage 2: Selecting a Goal (Goal Setup & Run)

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, Atrenta Console 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
```

Stage 2: Selecting a Goal (Goal Setup & Run)

```
#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

Atrenta Console 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, Atrenta Console ignores the `-host_config_file` command, as parallel goal run is not enabled. In addition, Atrenta Console 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, Atrenta Console ignores the `-host_config_file` command as parallel goal run is not enabled. In addition, Atrenta Console 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, Atrenta Console 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.

Stage 2: Selecting a Goal (Goal Setup & Run)

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. 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, Atrenta Console 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.

Central Design Setup

Most types of analysis in SpyGlass require additional design intent information to that provided by the design HDL and technology libraries. You can provide such information by using the setup wizard under the *Central Setup* tab.

The setup wizard provides you step-by-step guidance to supply additional design intent information. Most of this information is present in the form of SpyGlass design constraints saved in .sgdc files. Using this wizard, you can:

- Specify black boxes, clocks, and resets information that can be later used with the goal runs.
- Specify technology-specific design intent information by using various goal-specific wizards.

Follow the steps in the setup wizard as completely as possible. If a step requires information that is not readily available, you can skip that step initially. Such steps appear later in the goal-specific setup wizards, when required.

Stage 2: Selecting a Goal (Goal Setup & Run)

When you click on the *Central Setup* tab, the initial page of the setup wizard appears, as shown in the following figure:

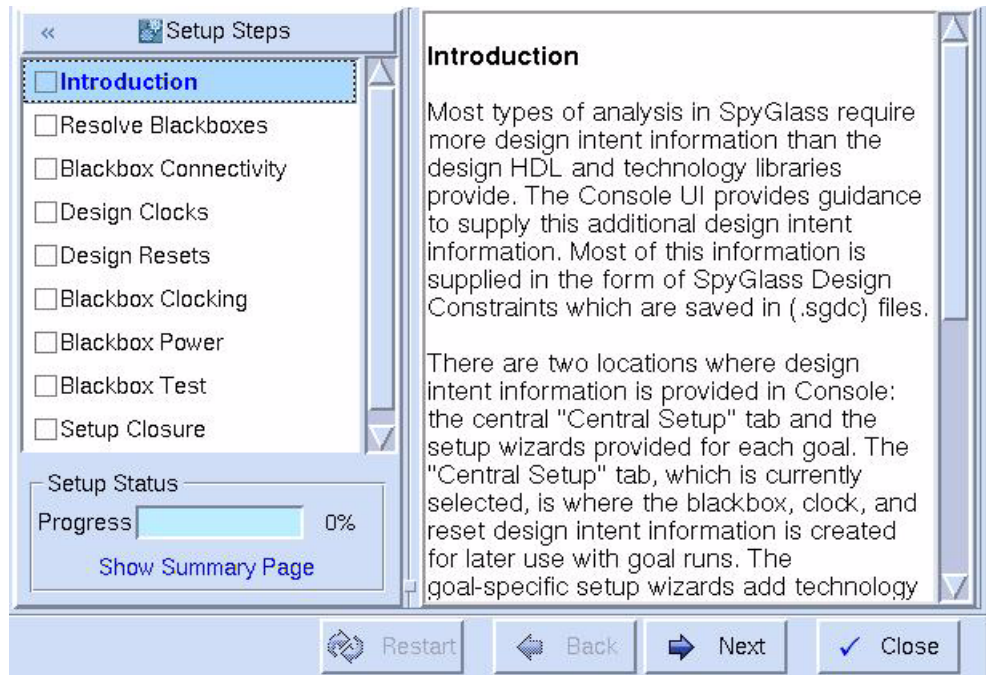


FIGURE 24. Central Design Setup - Introduction

In the above figure, the left pane displays an index of steps of the *Central Setup* wizard, and the right pane displays the description or instructions for that step. You can navigate through this wizard either by clicking on the steps directly or by clicking the *Back/Next* buttons.

NOTE: Some steps have prerequisite steps, and the wizard warns you if you do not follow those prerequisite steps.

Following is an overview of the steps of this setup wizard:

1. *Resolve Blackboxes* step

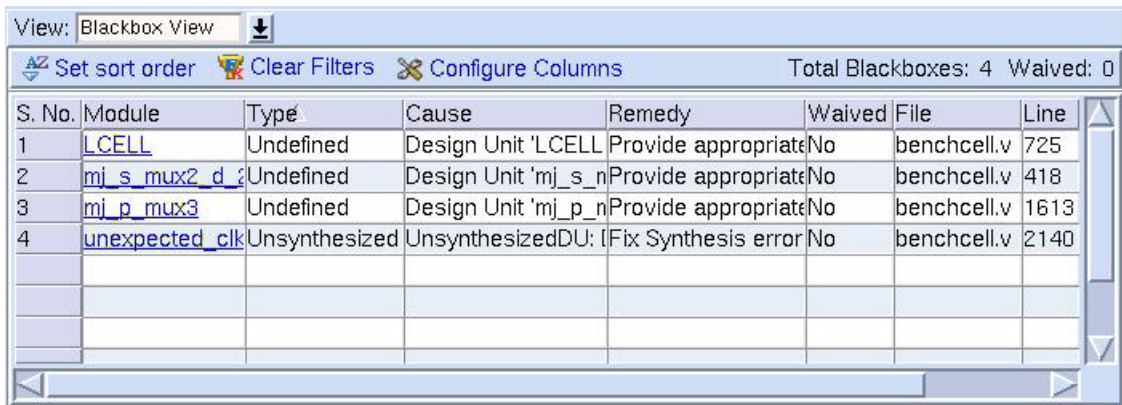
SpyGlass allows undefined modules to be left as black boxes. In many cases, however, the analysis remains incomplete if there are any unintentional black boxes in the design. Therefore, it is important to confirm that all black boxes are expected. To avoid unintentional black

boxes, this step finds all the black boxes through design synthesis and provides guidance on how a real model can be provided.

Click the *Next* button to run SpyGlass, and find all the black boxes in the design. If you have run SpyGlass earlier:

- Click the *Yes* button to run SpyGlass again or
- Click the *Show Last Results* button to view black boxes detected in the last run.

After this, SpyGlass lists all the black boxes in a spreadsheet view, as shown in the following figure:



S. No.	Module	Type	Cause	Remedy	Waived	File	Line
1	LCELL	Undefined	Design Unit 'LCELL	Provide appropriate	No	benchcell.v	725
2	mj_s_mux2_d	Undefined	Design Unit 'mj_s_n	Provide appropriate	No	benchcell.v	418
3	mj_p_mux3	Undefined	Design Unit 'mj_p_n	Provide appropriate	No	benchcell.v	1613
4	unexpected_clk	Unsynthesized	UnsynthesizedDU: [Fix Synthesis error	No	benchcell.v	2140

FIGURE 25. List of Blackboxes

In the above spreadsheet, black boxes information appear with type, cause, and potential remedy, in addition to other details.

To resolve or waive a black box, right-click on the black box name appearing in the *Module* field, and select the appropriate option from the shortcut menu.

2. *Blackbox Connectivity* step

This step models how black boxes are logically connected with the rest of the design.

The following figure shows the page appearing during this step:

Stage 2: Selecting a Goal (Goal Setup & Run)

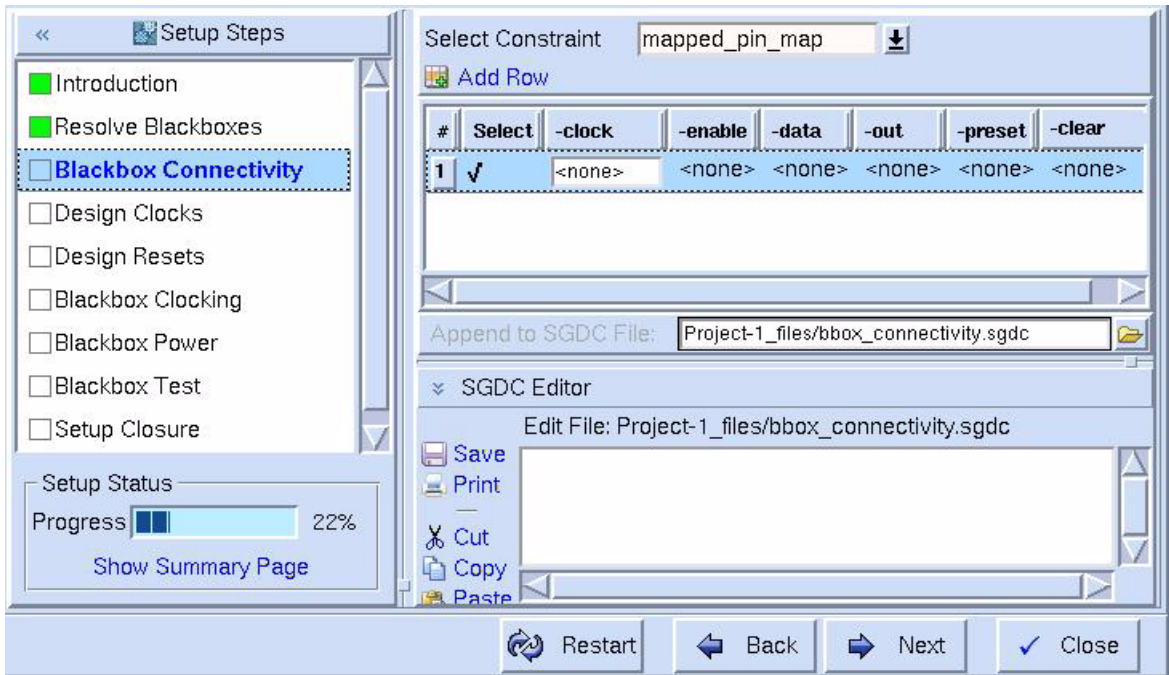



FIGURE 26. Central Design Setup - Blackbox Connectivity

In the above page, you can create two types of constraints, `assume_path` and `mapped_pin_map`, for each black box. You can refer to the *HTML Help* window that guides you in generating these constraints for black boxes.

This page also contains the *SGDC Editor* dialog in which you can write the constraint specifications for a black box. In addition, you can display an existing SGDC file in this dialog by clicking the  button to select an SGDC file in the *Append to SGDC File* section. Once the required SGDC file is loaded in the *SGDC Editor*, you can make the required modifications in that file.

NOTE: You can use the SGDC files generated in this step for other goal setup steps.

3. Design Clocks step

This step enables you to provide SGDC files or enable auto detection of clocks. Atrenta Console displays the following screen during this step:

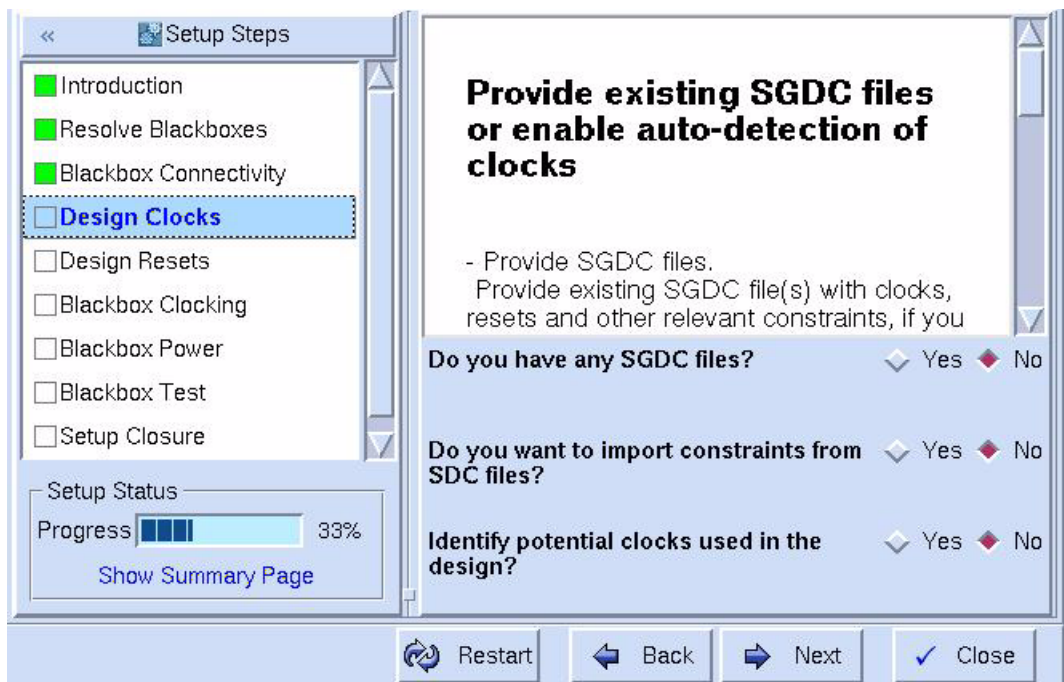


FIGURE 27. Central Design Setup - Design Clocks

In the above page, you can perform the following:

- Provide SGDC files with clocks, resets, and other relevant constraints, if you have any.

To provide SGDC files, select the *Yes* option corresponding to the *Do you have any SGDC files* label.

NOTE: You can use constraints created for other SpyGlass products.

- Provide sdc_data constraint to import SDC files

To provide existing SDC files, select the *Yes* option corresponding to the *Do you want to import constraints from SDC files* label.

The following example provides the sdc_data constraint in an SGDC file:

```
sdc_data -file "sdcfile.sdc"
```

Stage 2: Selecting a Goal (Goal Setup & Run)

NOTE: SDC file specified in the `sdc_data` constraint must exist.

Identify potential clocks in the design

To find all the clocks in the design, select the *Yes* option corresponding to the *Identify potential clocks used in the design* label. However, if you have already provided a constraint file and do not want SpyGlass to find more clocks, select the *No* option.

After selecting the required options, click the *Next* button. If you have selected the *Yes* option corresponding to the *Do you have any SGDC files* and/or *Do you want to import constraints from SDC files* labels, Atrenta Console displays the following screen:

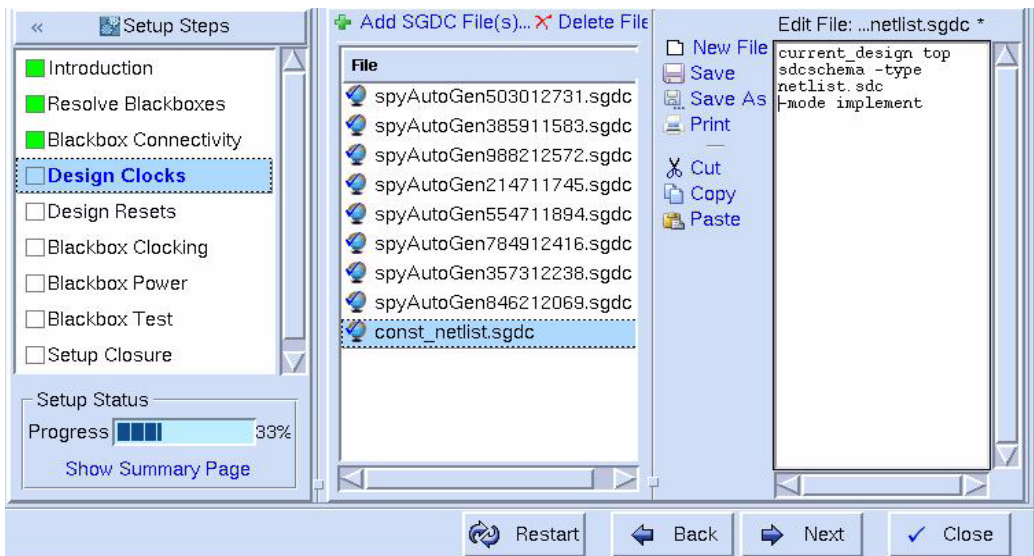


FIGURE 28. Central Design Setup - Importing Constraints

Here, Atrenta Console displays a list of the SGDC files that you have added during the *Design Setup* stage. You can append more SGDC files to this list by clicking the *Add SGDC File(s)* link.

When you select any SGDC file in this list, Atrenta Console displays the contents of that SGDC file in the *Edit File* section. You can then make the required modifications for that SGDC file, and save the changes. Atrenta

Console provides various options (such as *New File*, *Save*, *Save As*, etc.) on the left side of the *Edit File* section, in which you can perform various actions.

Once you have specified the SGDC files, click the *Next* button. This displays a screen in which you can analyze the clock trees, and tune the clock definitions. You can skip this step by clicking the *Skip* button.

However, to proceed with this step, click the *Yes* button. When you click the *Yes* button, Atrenta Console displays the results, as shown in the following figure:

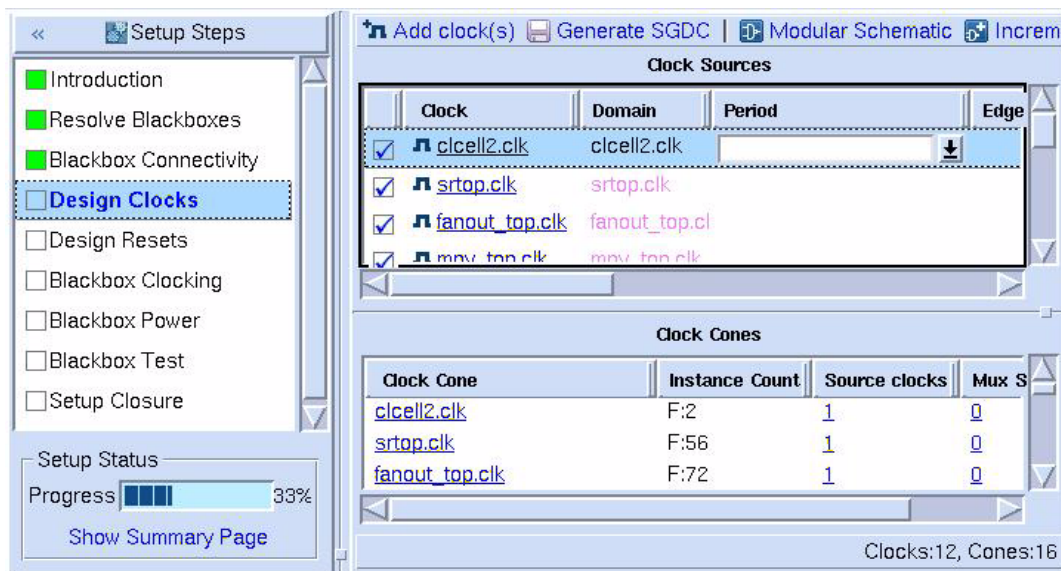


FIGURE 29. Central Design Setup - Analyzing Clocks

In the above setup page, click on a clock in the *Clock Sources* section to view all its cones in the *Clock Cones* section. You can also click on a cone in the *Clock Cones* section to view all its clock drivers in the *Clock Sources* section.

NOTE: *Clock cone* is the net in the path that is driving clock pin of sequential element(s).

Stage 2: Selecting a Goal (Goal Setup & Run)

In the *Instance Count* field under the *Clock Cones* section, the following convention is used to denote different types of objects:

F	Flip-flops
B	Black boxes
L	Latches
C	Sequential cells in the clock cone path

Ensure that you remove improper clocks, add missing clocks, put synchronous clocks into the same domain, set the correct frequencies, mark test clocks, and save the final clock information in the SGDC file. You can specify missing constraints in clock path.

4. *Design Resets* step

This step enables you to set up asynchronous resets in the design.

You can skip this step by clicking the *Skip* button. However, to proceed with this step, click the *Yes* button. When you click the *Yes* button, Atrenta Console displays the following screen:

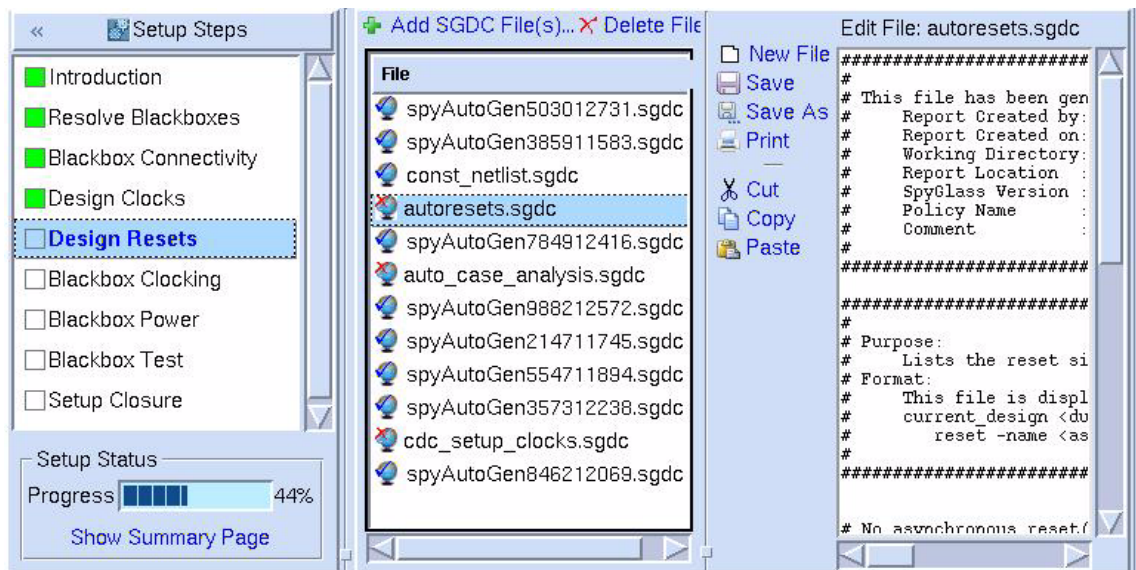


FIGURE 30. Central Design Setup - Design Resets

SpyGlass extracts asynchronous resets of the design in the `autoresets.sgdc` file. Here, you can review, edit, and finalize the reset constraints.

5. *Blackbox Clocking* step

This step enables you to model clock domains for black boxes that have more than one clock pins. The following page appears during this step:

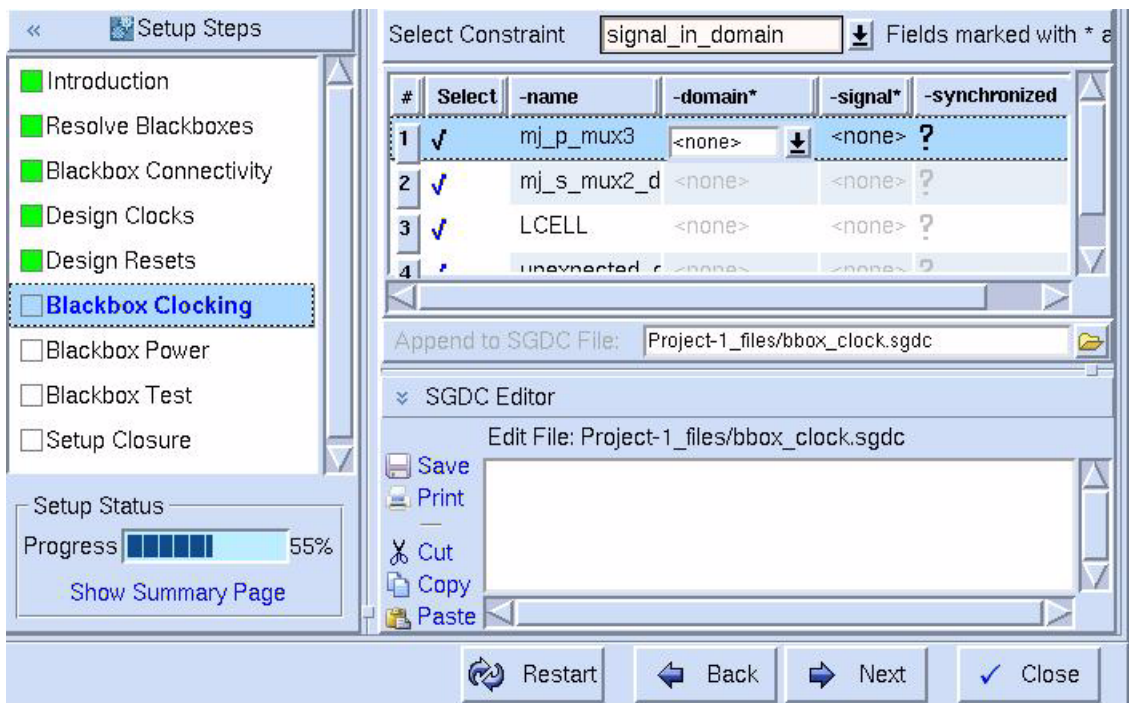


FIGURE 31. Central Design Setup - Blackbox Clocking

Modeling a black box is necessary for advanced clocking and Clock Domain Crossing (CDC) analysis. Run this setup step only if test analysis will run. Otherwise, you may skip this step.

In this step, you can refer to the *Help* section to see the steps to model black boxes.

6. *Blackbox Power* step

This step enables you to specify the constraints for black box power.

Stage 2: Selecting a Goal (Goal Setup & Run)

This step is similar to the previous step in which you can select the required constraint from the *Select Constraint* drop-down list, and then specify the values for various arguments of the selected constraint.

The following page appears during this step:

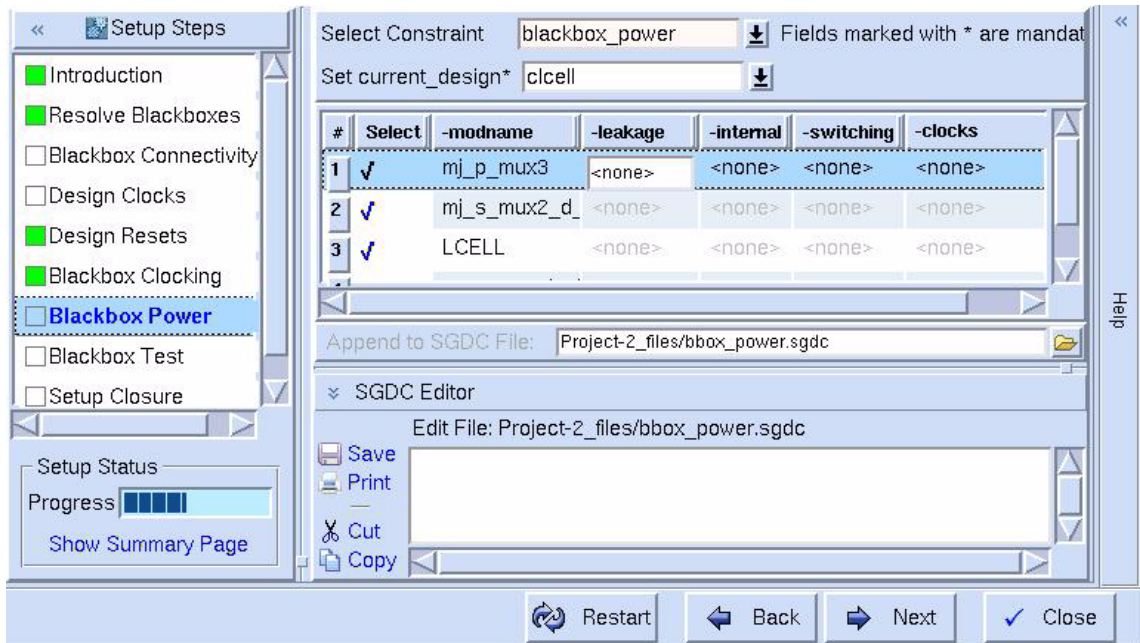


FIGURE 32. Central Design Setup - Blackbox Power

After specifying the required constraints for black box power, click the *Next* button.

7. *Blackbox Test* step

This step enables you to model a black box for test analysis. Run this setup step only if test analysis will run. Otherwise, you may skip this step.

The following page appears during this step:

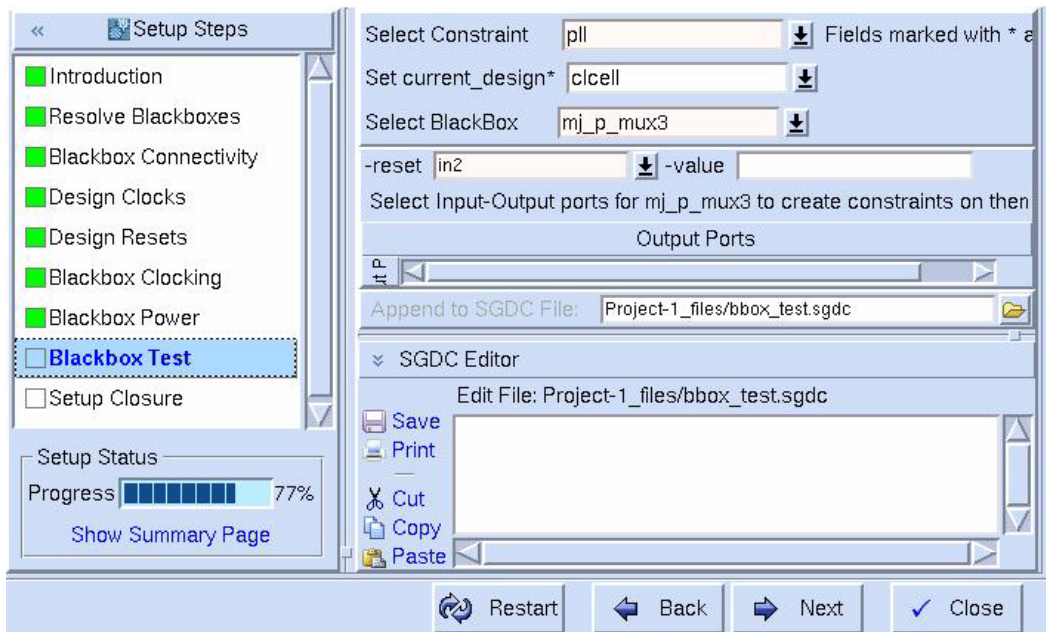


FIGURE 33. Central Design Setup- Blackbox Test

Modeling a black box for test analysis involves categorizing the following black box types:

- Clock generators
- Clock gating cells
- Resistive cells
- Other black boxes

Each of the above black box type requires different type of constraints. Since these types are mutually exclusive, they appear in four different screens.

You can refer to the help in the *Help* section to see the steps for modeling each type of black box.

8. *Setup Closure* step

This step enables you to run a sanity check to check if you have specified constraints properly. You can skip this step by clicking the *No*

Stage 2: Selecting a Goal (Goal Setup & Run)

button. However, if you want to proceed with this step, click the *Yes* button.

After successfully completing the setup closure step, click the *Finish* button to close this wizard.

Now, the design has been setup for black boxes, and you can now review the SGDC files and go to the *Select Goal* step to setup or run the other goal(s).

Setting up the Goal

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

Determining Parameter Precedence

A particular parameter may be assigned a different value in the goal file, a project file, and while setting up a goal. In such cases, Atrenta Console gives precedence to parameter value in the following order (starting from the highest priority)

- Parameter value set during the goal setup stage
- Parameter value specified in the project file
- Parameter value specified in the goal file (.spq)

Setting Parameters and Constraints for Selected Goal

To set the parameters and constraints for the selected goal(s), click the *Setup Goal* tab.

NOTE: *The Setup Goal tab is enabled only when the anchor (the blue bar) is on a goal.*

Atrenta Console provides guidance in the steps for setting up goals for which the status appears as *Setup Recommended*. The setup is done through the <goal-name>_setup.sgs file. This file is present in the same directory as that of the goal in the methodology directory hierarchy. It is recommended that whenever you copy methodologies, either from *Methodology Configuration System* window or manually, you should also copy .sgs files for

the goal setup steps to run.

To use the wizard to set up a goal, perform the following steps:

1. Select a goal that requires additional setup steps and click the *Setup Goal* tab.

The following setup wizard appears:

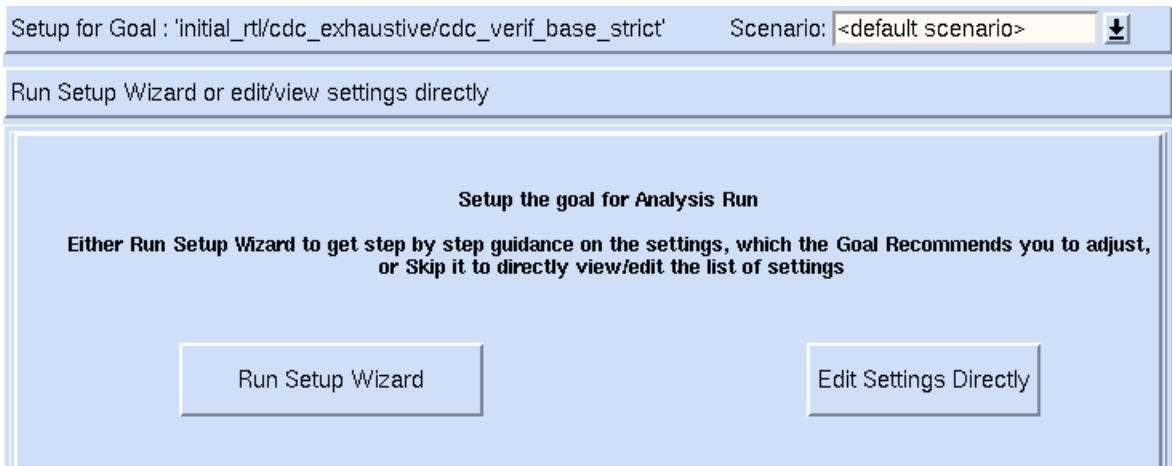


FIGURE 34. Setup Goals

NOTE: You can click the *Edit Settings Directly* button to skip the wizard.

2. Click *Run Setup Wizard* button to run the wizard. The Setup Summary page appears, as shown below.

Stage 2: Selecting a Goal (Goal Setup & Run)

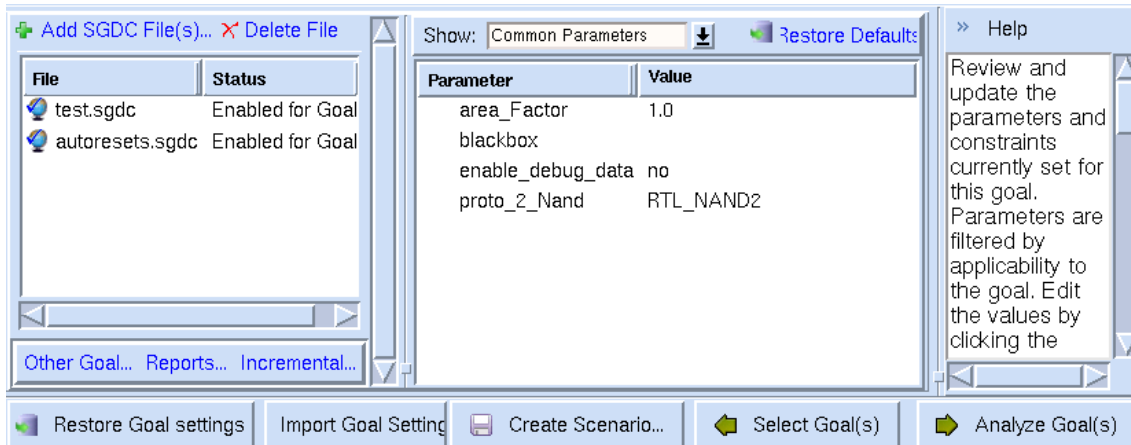


FIGURE 35. Setup Wizard - Setup Summary

The Setup Summary page is divided into the following three sections:

- ❑ The left section displays a sample SGDC file and the recommended constraints for the selected goals appear under the *Recommended Constraints*. When you left-click, the *help* link located next to a constraint the help related to the constraint appears in the right section of the Constraints page.

You can enable or disable a constraint by selecting the *Enabled for Goal* or *Disabled for Goal* option, respectively, from the drop-down list in the *Status* field. You can also toggle goal status of the SGDC file by right-clicking the file name and selecting the *Globally Enable File* or *Globally Disable File* option from the shortcut menu.

To add a constraints file, perform the following steps:

- a. Click the *Add SGDC File(s)* link.

The *Add File(s)* dialog appears, as shown in the following figure:

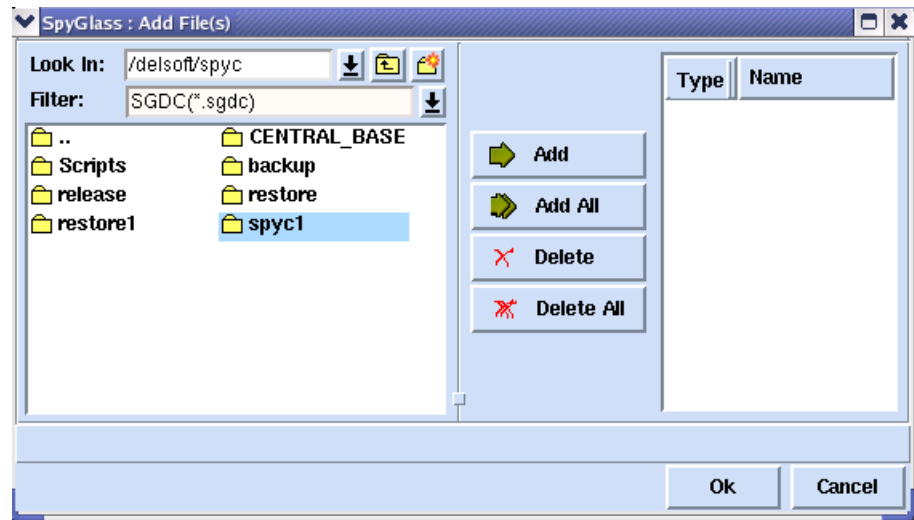


FIGURE 36. Setup Wizard - Add File(s)

- b. Browse to the directory that contains the SGDC files and select the required SGDC file.
- c. Click the *Add* button.

To add all SGDC files, click the *Add All* button. The selected SGDC files are appear in the Setup Summary page.

Alternatively, right-click on the *Setup Summary* page and select the *Add Constraint File(s)* option from the shortcut menu to add a constraints file.

NOTE: Whenever a new constraint file is added during the goal setup stage, that file is considered as local, that is, enabled for the current goal only. To change the status of this file to global, right-click the file name and select the *Globally Enable File* option from the shortcut menu.

To delete a constraints file, select the constraints file and click *Delete File* link. Alternatively, right-click on the constraints file and select the *Delete* shortcut menu option or click the <Delete> key on the keyboard.

The leftmost pane also contains a sub section, *Other Goal Command Line Options*. In this section, you can provide the goal specific options that are applicable to the scope of a specific goal.

Stage 2: Selecting a Goal (Goal Setup & Run)

- ❑ The middle section displays the source of the .sgdc file. You can edit the file and click *Save* to save the file.

To create a constraints file, right-click the *New File* link. The *Open New File* dialog appears, in which you can specify the name of the file. After specifying the required details, click the *Open* button. The new file appears in the left section of the *Constraints* window. Next, you can enter the code for the new constraints file and click *Save* to save the file. You can also click the *Save As* link to save the file to a different location.

You can also use the *Cut*, *Copy*, and *Paste* links to copy the text in the Constraints to any text editor.

- ❑ The right section displays the help related to the constraint displayed under *Recommended Constraints*.

3. After adding the constraints file, click *Finish*. To move back to the previous screen of the Setup wizard, click *Back*. You can also click the *Restart* button to go back the first step of the Setup wizard.

When you click the *Restart* button, Atrenta Console displays a dialog that prompts you to confirm whether you want to reset the settings for the current goal. In this dialog, click the *Yes* button to clear the settings for the selected goal and go back to the first step. Click the *No* button to go back to the first step with the current settings.

NOTE: You can click the *Close* button at any step to proceed directly to the *Constraints Summary* page.

When you set a goal by using the Setup Wizard (or click *Edit Settings Directly* button), the *Constraints Summary* page appears, as shown in the following figure.

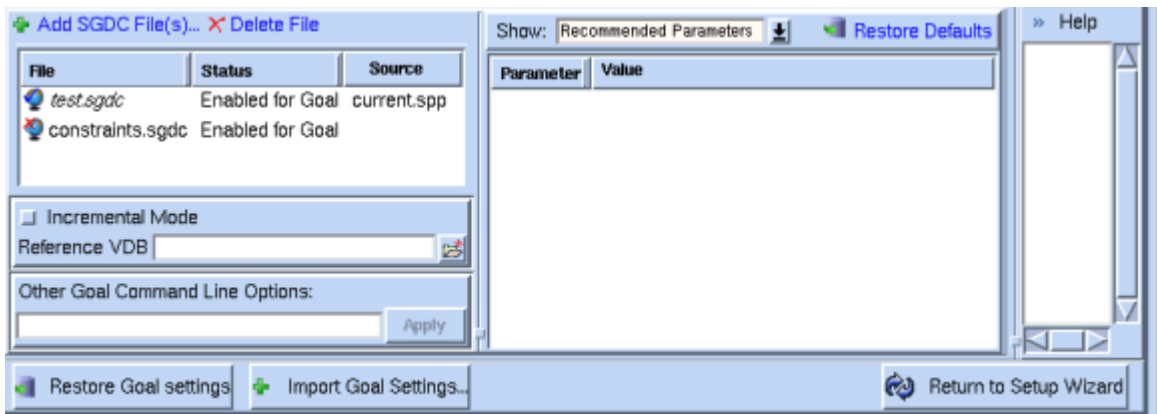


FIGURE 37. Setup Wizard - Constraints Summary

The Constraints Summary page displays the SGDC files that you have included for analyzing your designs for the selected goal.

Like the Setup Summary page, the Constraints Summary page is divided into three sections as follows:

- The left section displays the selected constraints. If you have not used the Setup wizard to specify the constraints, you can add the constraints in this section by clicking the Add SGDC File(s) link or by right-clicking anywhere on the section and selecting the *Add Constraint File(s)...* shortcut menu option.
- The middle section displays the parameters that you need to set for the selected goal. By default, this section displays the recommended parameters that you need to set for the selected goal. You can, however, view the common parameter list by clicking the *Show* drop-down list and selecting the *Common Parameters* option. In addition, you can view the complete list of parameters that can be set for the selected goal by selecting the *All parameters* option from the Show drop-down menu.

NOTE: *The Recommended Parameters option is not available for goals that do not require additional setup steps.*

- When you select a parameter, the right section displays the help related to that parameter.

You can click the *Restore Goal settings* button to deselect the constraints,

Stage 2: Selecting a Goal (Goal Setup & Run)

restore the parameters to their default values, and restore all the settings of the current goal.

For a goal that requires additional setup, you can copy the settings (constraints, parameters, and other settings) from another goal that has a similar setup so that the settings can be reused. To do this, click the *Import Goal Settings* button. Then the *Select the goal whose setting you wish to import* dialog appears as shown in the following figure:

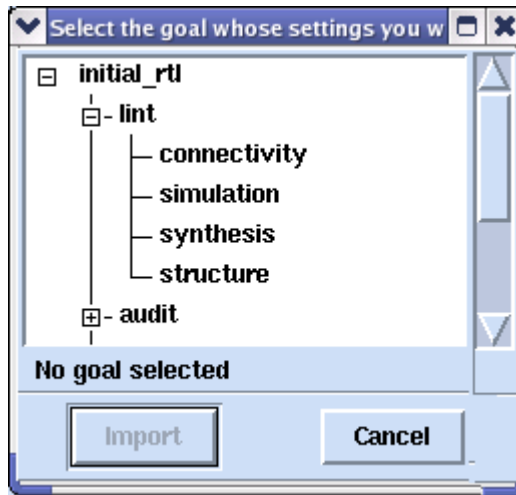


FIGURE 38. Setup Wizard - Import Goal Settings

Select the required goal whose setting you want to import and click *Import*.

You can click the *Return to Setup Wizard* button to navigate to the first step of the Setup Wizard.

NOTE: *This button is visible only if you have set the constraints using the Setup Wizard.*

During the *Goal Setup* closure step, you can select the SGDC files created in the previous steps, and pass them to the current goal run, or you can create a top-level SGDC file to be used in some other project. You can do this by using the following screen of the setup closure step:

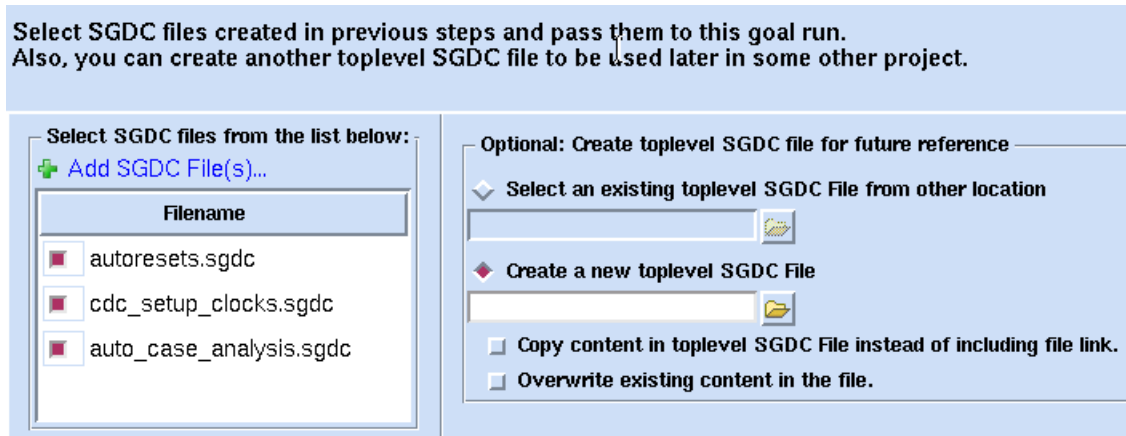


FIGURE 39. Setup Wizard - Add SGDC File(s)

In the above screen, the *Filename* section displays a list of SGDC files. If you click the *Next* button, the selected SGDC files in this section are globally enabled.

Now, to specify an SGDC file in which you can add the contents of the selected SGDC files appearing in the *Filename* section, select either of the following options:

- Select an existing top-level SGDC file from other location
Select this option to load an existing top-level SGDC file.
- *Create a new toplevel SGDC File* option.
Select this option to create a new top-level SGDC file.

Once you have specified an SGDC file, you can select the following options:

- *Copy content in top level SHDC File instead of including the file link*
Select this option to copy the contents of the selected SGDC files, appearing in the *Filename* section, in the specified SGDC file.
- *Overwrite existing content in the file*
Select this option to overwrite the contents of the specified SGDC file with the contents of the selected SGDC files appearing in the *Filename* section.

If you do not select any of the above options, Atrenta Console adds a link

of the selected SGDC files in the specified file.

Performing Sanity Checks for Parameters

Atrenta Console performs sanity checks on parameter values to indicate an invalid parameter value. If you enter a wrong value for a parameter, the color of that parameter changes to red. Placing the mouse pointer over that parameter displays a balloon help window that contains a brief description about the valid values that can be entered for the rule parameter. These checks are applicable only for rule parameters that do not take design related inputs, such as integers, floating values, alphanumeric strings, and so on.

NOTE: *Currently, only the SpyGlass Base products, SpyGlass CDC solution, and the SpyGlass DFT solution have implemented this check. The parameter sanity checking will be available for other products in a future SpyGlass release.*

Using the Dual Design Read (DDR) Flow

The DDR flow is a capability of reading two designs simultaneously in a single SpyGlass run, and perform a variety of comparative analysis on these designs.

The first design is called the reference design as it is the golden input, and the second design is called the implementation design as it is the design under analysis.

This flow is used by the following features in SpyGlass:

■ SDC Equivalence Analysis

It is used to compare two different SDC files for pre-layout and post-layout design snapshots.

■ Sequential Equivalence Checking (SEC)

It is used to sequentially compare the original design and its corresponding power-optimized design to ensure that both the designs are functionally equivalent.

You can implement the DDR by using the setup wizard under the *Setup Goal* tab. This setup assumes that you have already read in the implement design during design read. In this setup, you can set up the SGDC file for the implement design, and then specify the reference design against which

the equivalence needs to be done.

When you click the *Setup Goal* tab, the following screen appears:

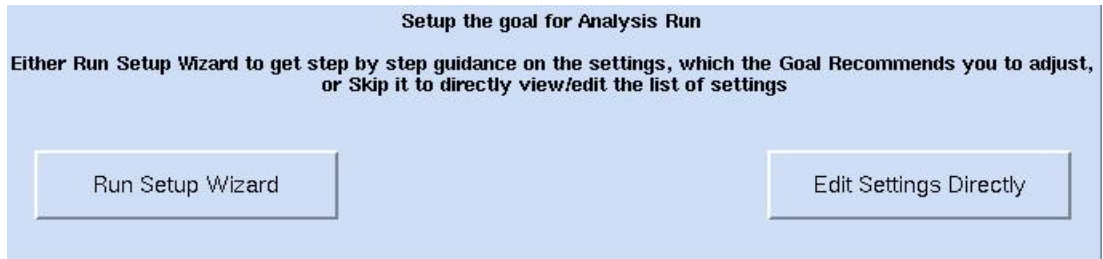


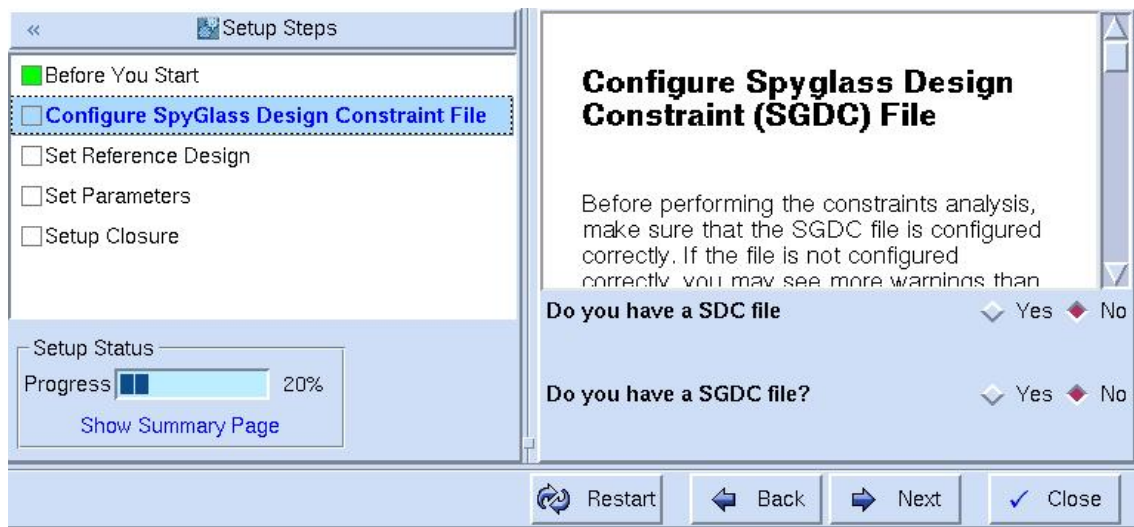
FIGURE 40. Setup Wizard - Goal Setup for Analysis Run

In the above screen, click the *Run Setup Wizard* button. This displays the first screen of the setup wizard that provides a brief overview of SDC Equivalence Dual Design. Click the *Next* button to proceed to the next step. The details of the subsequent steps of this setup wizard are discussed below:

1. *Configure SpyGlass Design Constraint File* step

This step enables you to configure your SGDC file correctly. During this step, the following screen appears:

Stage 2: Selecting a Goal (Goal Setup & Run)

**FIGURE 41.** Setup Wizard - Configure SGDC File

If you want to specify SDC and/or SGDC file, select the *Yes* option corresponding to the labels, *Do you have SDC file* and/or *Do you have a SGDC file*, respectively. After selecting the *Yes* option, click the *Next* button. This displays a screen containing a textbox in which you can specify the required file. Once you have specified the required file, click the *Next* button. You can also skip this step without specifying any file by directly clicking the *Next* button.

When you click the *Next* button, the following screen appears:

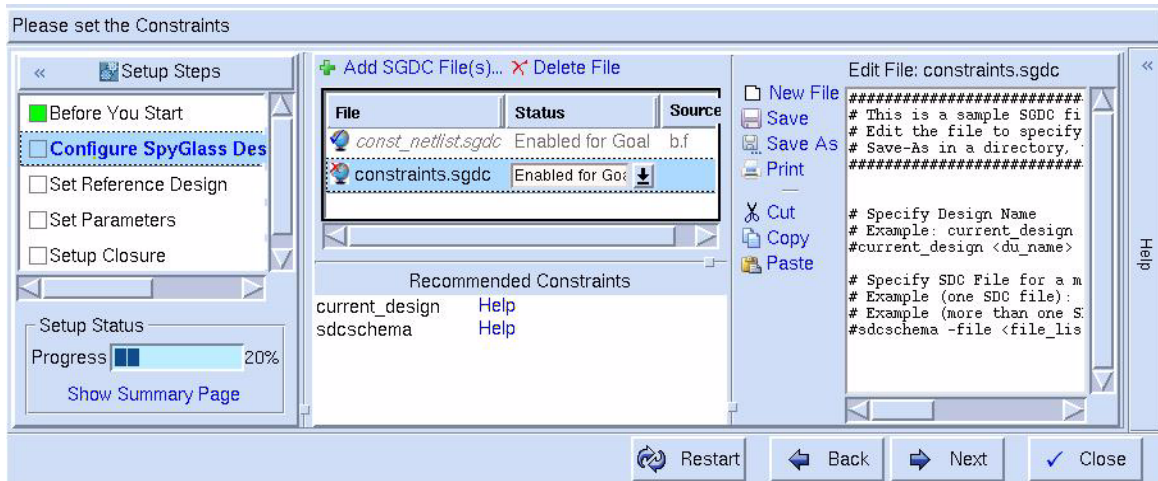


FIGURE 42. Setup Wizard - Setting the Constraints

In the above screen, you will set the SGDC constraints. Here, a list of SGDC files appear that were specified during the design read step. You can add more SGDC files to this list by clicking the *Add SGDC File(s)* link. By default, each SGDC file is enabled for the goal. You can disable an SGDC file for the goal by selecting the *Disable for Goal* option from the drop-down list appearing in the *Status* field.

When you select an SGDC file in this screen, contents of that file are displayed in the *Edit file* section. In this section, you can perform the required modifications in the SGDC file. On the left side of this section, Atrenta Console displays various links to perform different actions.

After performing the required actions, click the *Next* button to proceed to the next step.

2. *Set Reference Design* step

In this step, you provide the source file list of your reference design along with all the associated SGDC files. The first screen of this step displays an HTML help providing a brief introduction of this step. Click the *Next* button to proceed to the next screen, as shown in the following figure:

Stage 2: Selecting a Goal (Goal Setup & Run)

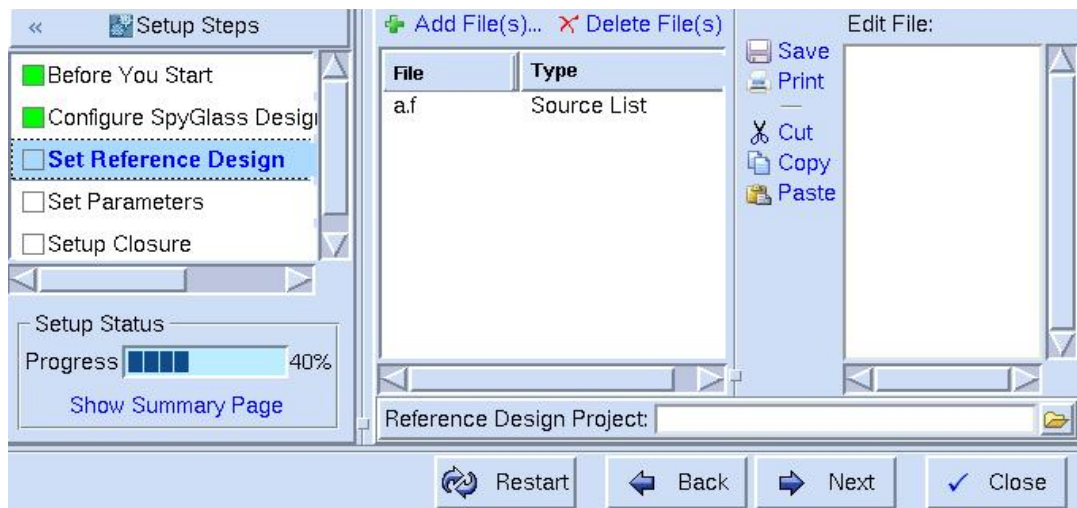


FIGURE 43. Setup Wizard - Set Reference Design

Here, click the *Add File(s)* link to add a reference design source list.

To modify a reference design source list, click on that source list file.

This displays the content of the file in the *Edit file* section. In this section, you can perform the required updates.

If the reference design was previously analyzed in Atrenta Console and the HDL files are listed explicitly, that project file can be added instead of the source list file. It is recommended to run the reference design through the design read process before being used with SDC Equivalence analysis. This will help ensure that the reference design itself is complete and lint-clean.

After specifying the reference design, click the *Next* button to proceed to the next step.

3. *Set Parameters* step

This step is used to generate the design equivalence file. This file maps ports, registers, and any intermediate design object used in the SDC files of one design to another. At the end of the run, you can review the generated file.

The following screen appears during this step:



FIGURE 44. Setup Wizard - Set Parameters

If you already have a design equivalence file that you want to use, select the *No* button in the above screen, and specify the file in the next step. However, if you want to generate the design equivalence file, click the *Yes* button.

After generating the design equivalence file, SpyGlass displays the results in the *Results* section, as shown in the following figure:

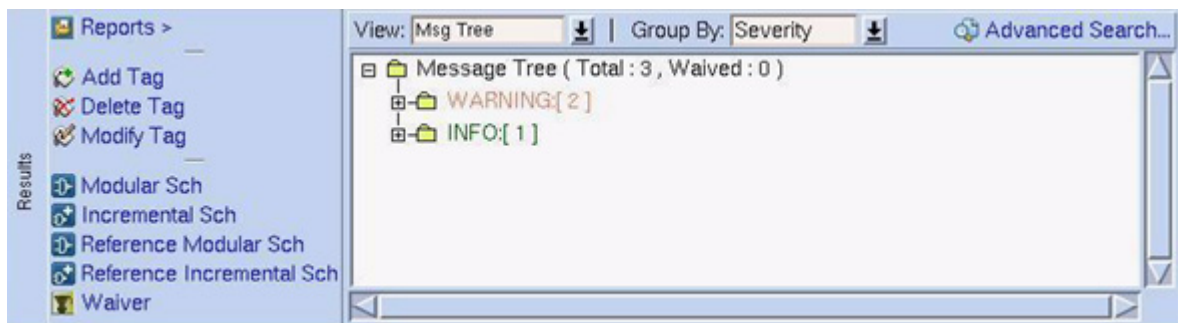


FIGURE 45. Setup Wizard - Generating Design Equivalence File

Stage 2: Selecting a Goal (Goal Setup & Run)

In the above figure, note the two extra links, *Reference Modular Sch* and *Reference Incremental Sch*. Click on these links to display the *Reference Schematic* and *Reference Modular Schematic* windows, respectively. These schematic windows enable you to compare the two given designs (implementation design and reference design). In addition, the Tools menu is changed to display the schematic options, *Implement Modular Schematic*, *Implement Incremental Schematic*, *Reference Modular Schematic*, and *Reference Incremental Schematic*.

Click the *Next* button to display a list of equivalent objects that are inherited from equivalence file and a list of equivalent objects that are found based on names, as shown in the following figure:

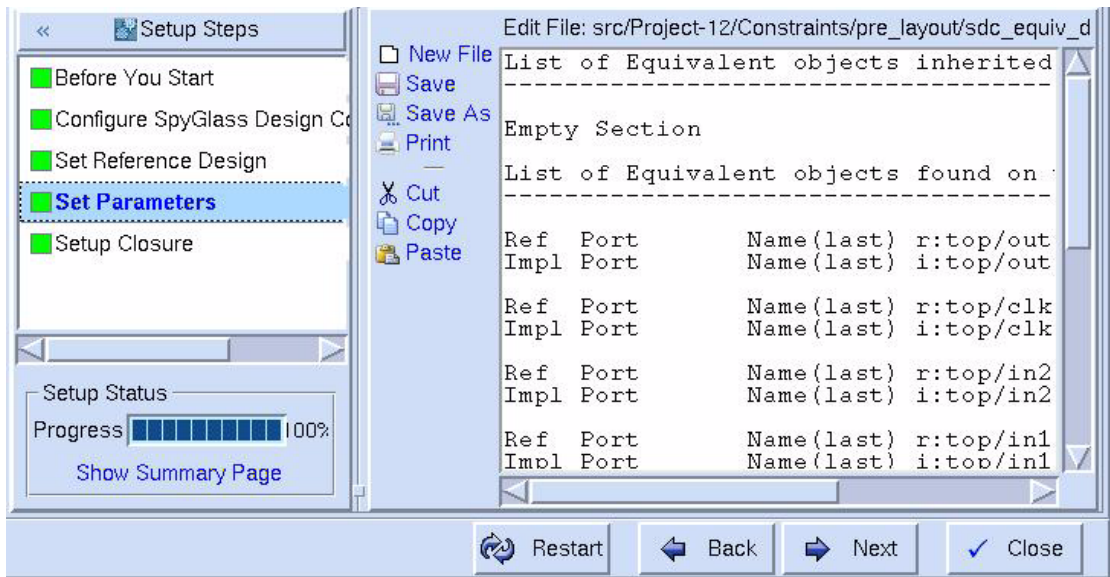


FIGURE 46. Setup Wizard - List of Equivalent Objects

Now, click the *Next* button.

4. *Setup Closure* step

In this step, click the Finish button to close the setup wizard.

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, Atrenta Console 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 Atrenta Console 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

 Stage 2: Selecting a Goal (Goal Setup & Run)

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

Atrenta Console 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:

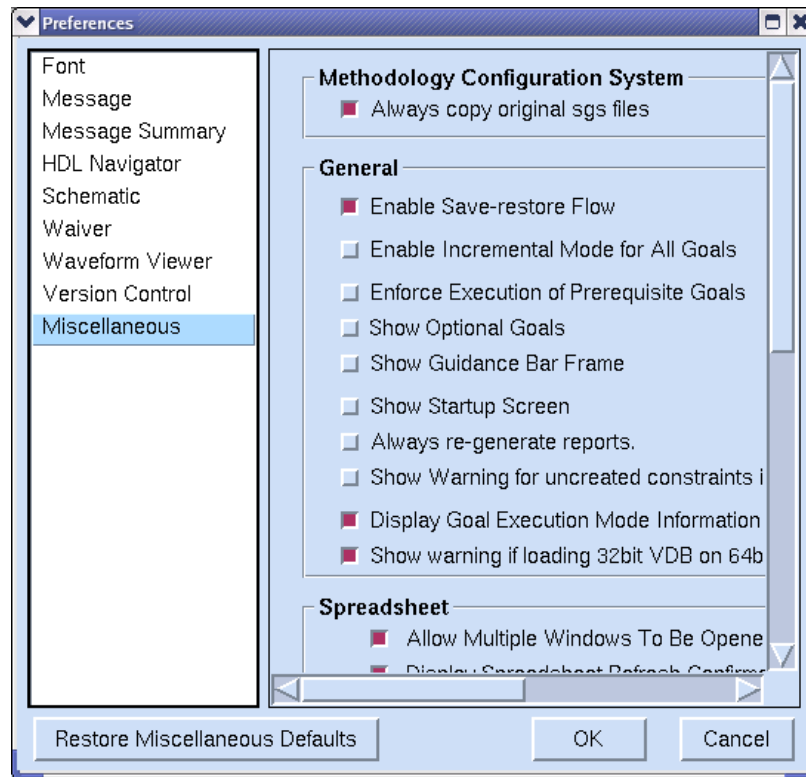


FIGURE 47. Methodology Configuration System - Preferences

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), Atrenta Console 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.

Stage 2: Selecting a Goal (Goal Setup & Run)

- *Do not uncheck 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:

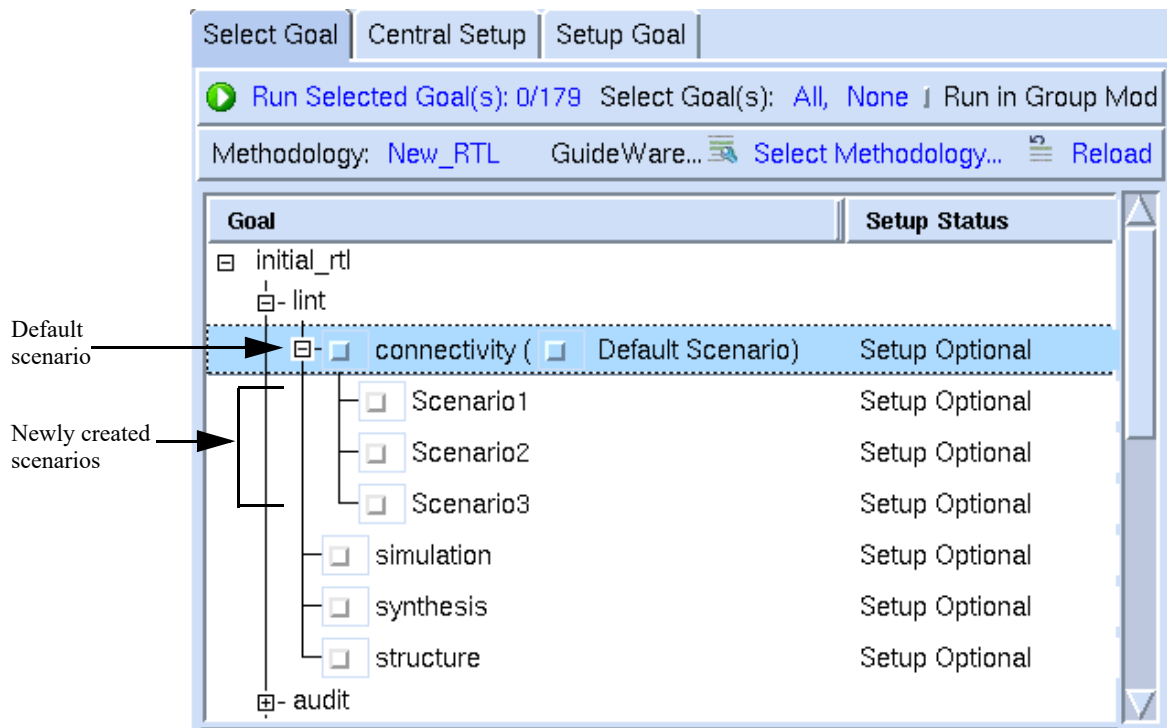


FIGURE 48. Run Selected Goal(s)

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.

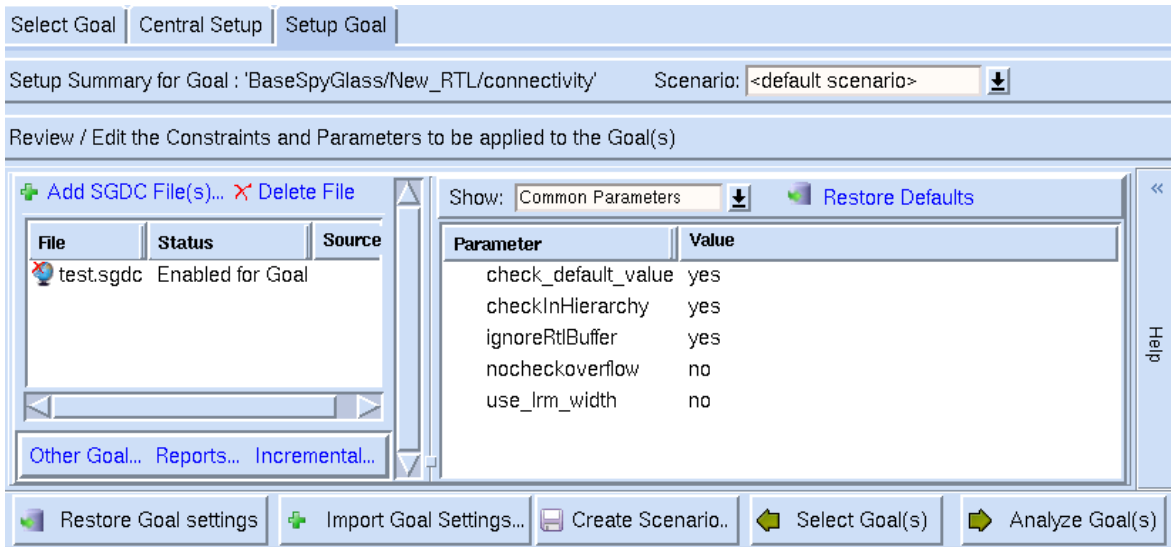
Creating Scenarios

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.

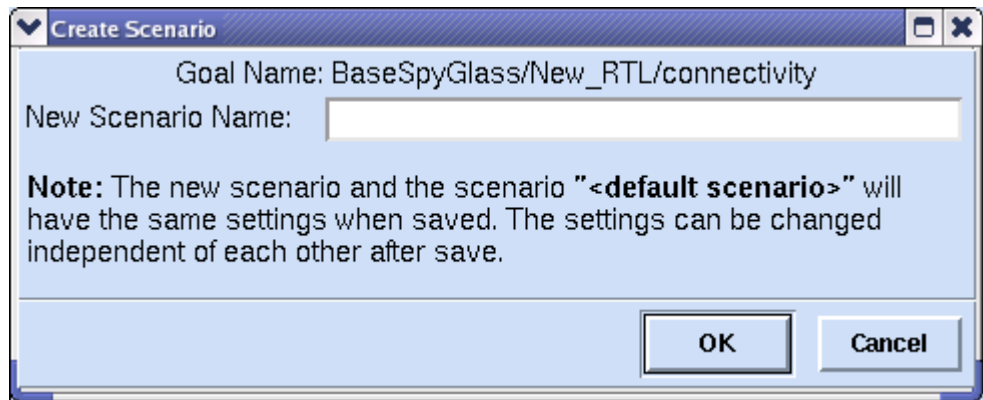
The page under the *Setup Goal* tab appears, as shown in the following figure:

Stage 2: Selecting a Goal (Goal Setup & Run)

**FIGURE 49.** Creating Scenarios

- Specify the required settings in the above page. For example, you can change parameter values or add/remove certain files.
- Click the *Create Scenario* button.

The *Create Scenario* dialog appears, as shown in the following figure:

**FIGURE 50.** 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:

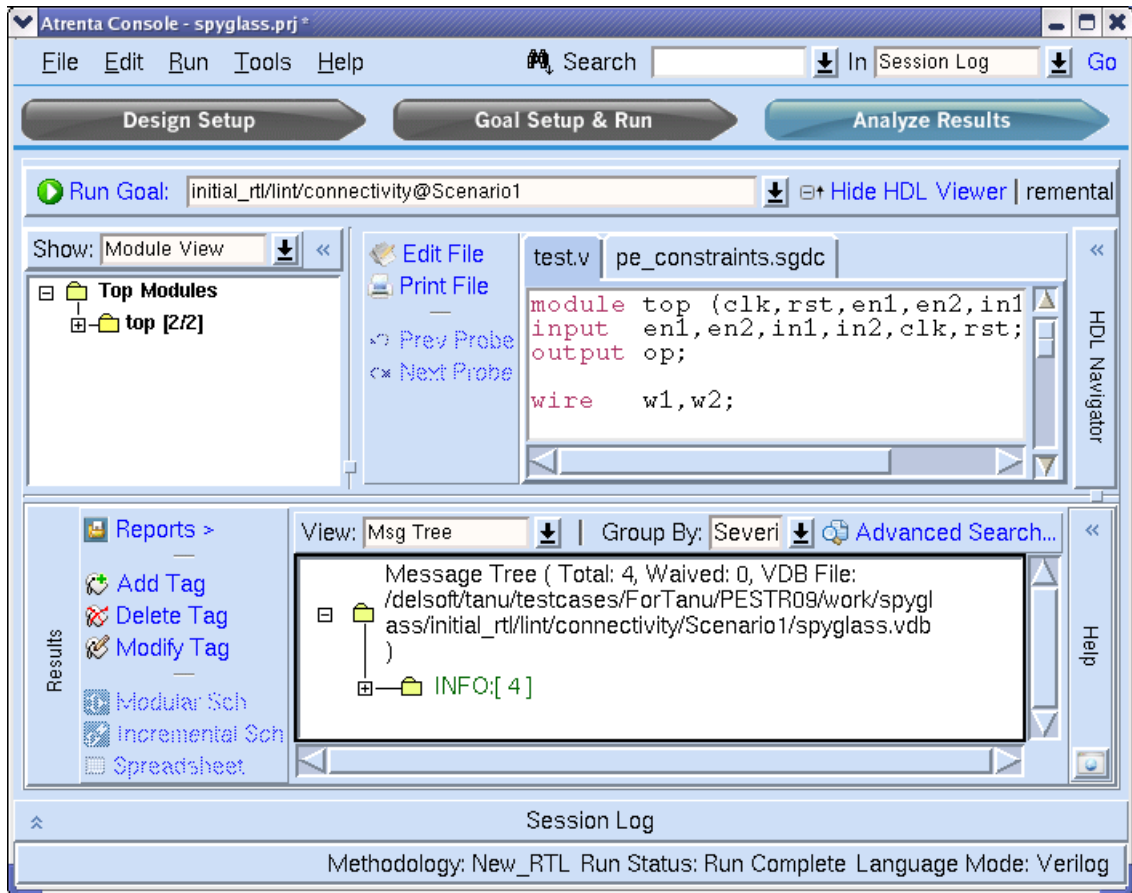


FIGURE 51. Analyze Results Tab

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.

Stage 3: Analyzing a Design (Analyze Results)


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

Atrenta Console 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, Atrenta Console 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:

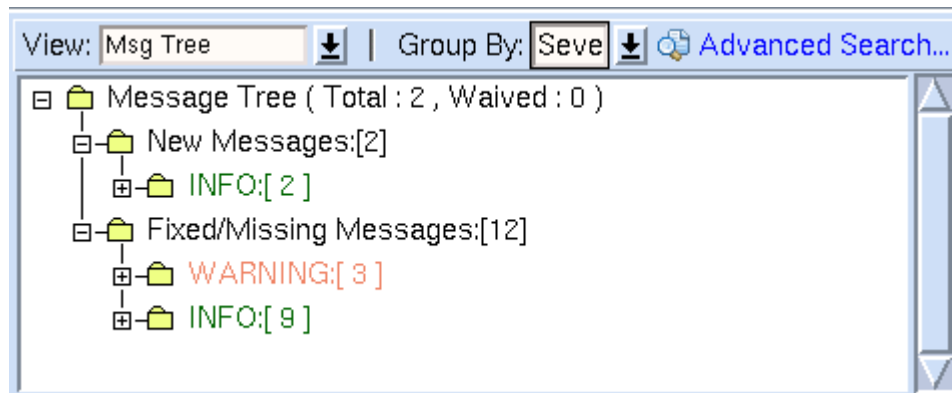


FIGURE 52. SpyGlass Violations

Using the vdb File

Atrenta Console 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.

Atrenta Console 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, Atrenta Console displays appropriate options to view different results, such as [Design Results](#), [SpyGlass CDC Solution Results](#),

Stage 3: Analyzing a Design (Analyze 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:

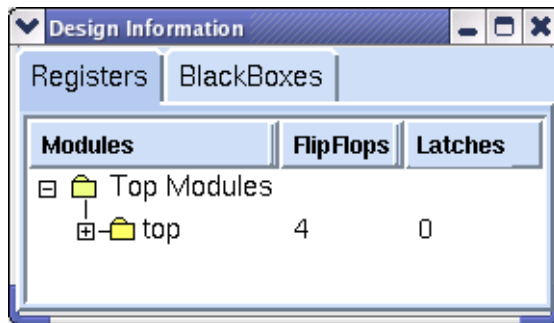


FIGURE 53. Design Information

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.

Refer to the *Viewing Blackbox Information* topic in *Atrenta Console Reference Guide* for details on viewing black box information.

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:

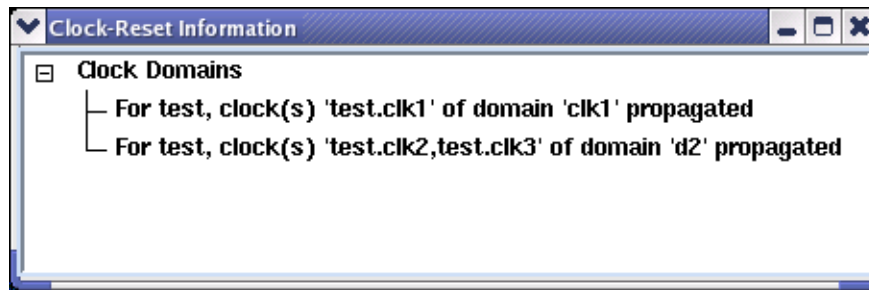


FIGURE 54. Clock-Reset Information

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

Stage 3: Analyzing a Design (Analyze Results)

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:

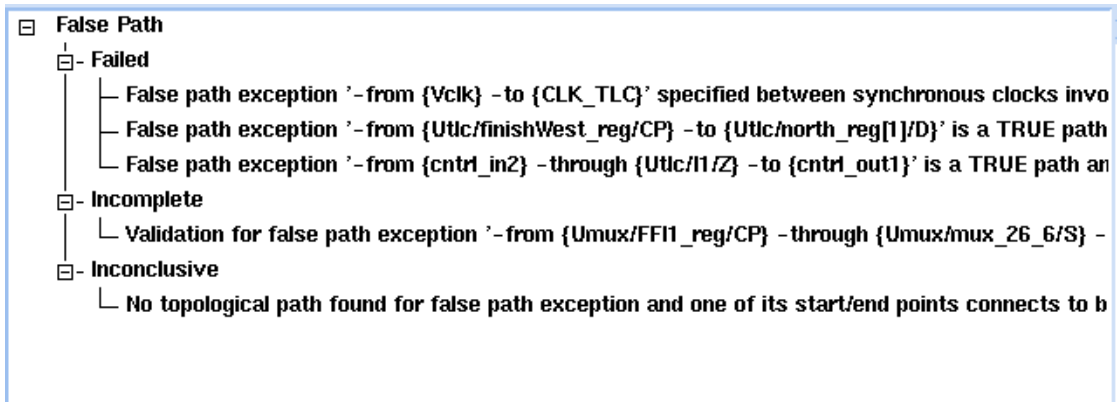


FIGURE 55. False Path Nodes

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:

Stage 3: Analyzing a Design (Analyze Results)

Instance	Module	Fault-Coverage	Test-Coverage
☐ SOCRATES	SOCRATES	N/A	N/A
☐ Utlc	tlc	N/A	N/A
☐ mcp_blk	multicycle_blk	N/A	N/A
☐ Umux	mux50by50	N/A	N/A
☐ RefDesCore	RefDesCore	N/A	N/A
☐ IO_spi	IO_spi	N/A	N/A
☐ IO_gpio	IO_gpio	N/A	N/A
☐ IO_usb	IO_usb	N/A	N/A
☐ IO_irda	IO_irda	N/A	N/A
☐ IO_eth	IO_eth	N/A	N/A
☐ IO_pci	IO_pci	N/A	N/A

Page 1 of 2 →

FIGURE 56. 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 or goal from the drop-down list, as shown in the following figure:

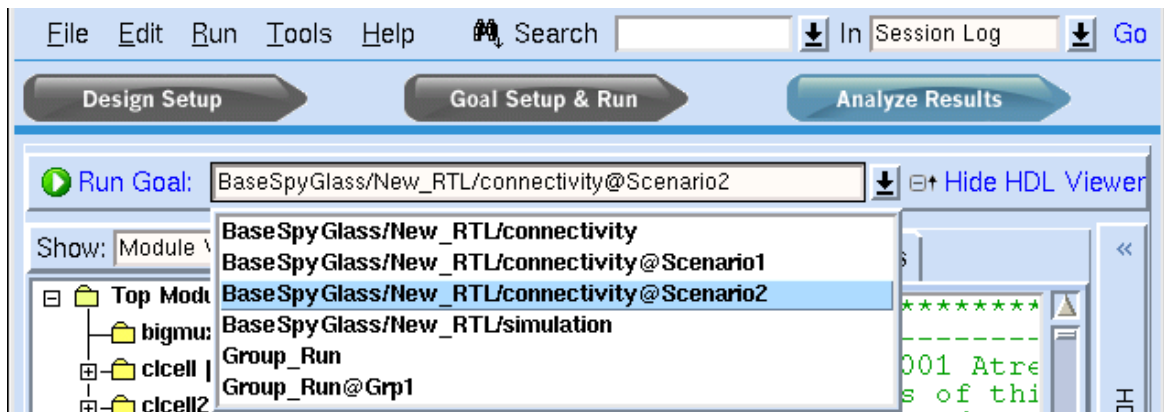


FIGURE 57. Analyze Results - Run Goal

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 Atrenta Console 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 Φ) is highlighted in the *Modular Schematic Window* and the *Incremental Schematic Window*.

Stage 3: Analyzing a Design (Analyze Results)

NOTE: *For more information, refer to the Atrenta Console Reference Guide.*

NOTE: *Atrenta Console 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.*

Working with Input Design and Libraries

Overview

This chapter describes all aspects of reading a design in Atrenta Console. 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, Atrenta Console 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:

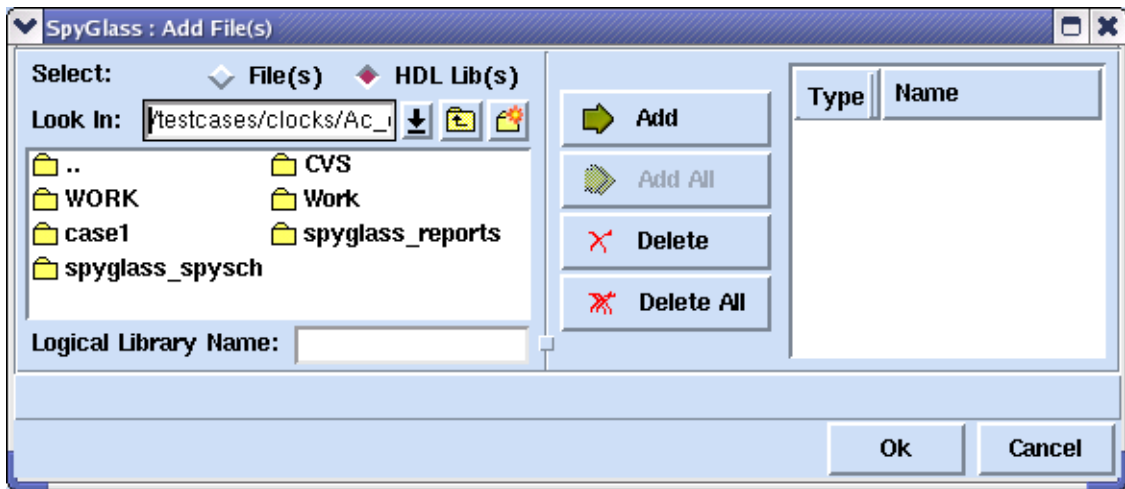



FIGURE 1. Add File(s)

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:

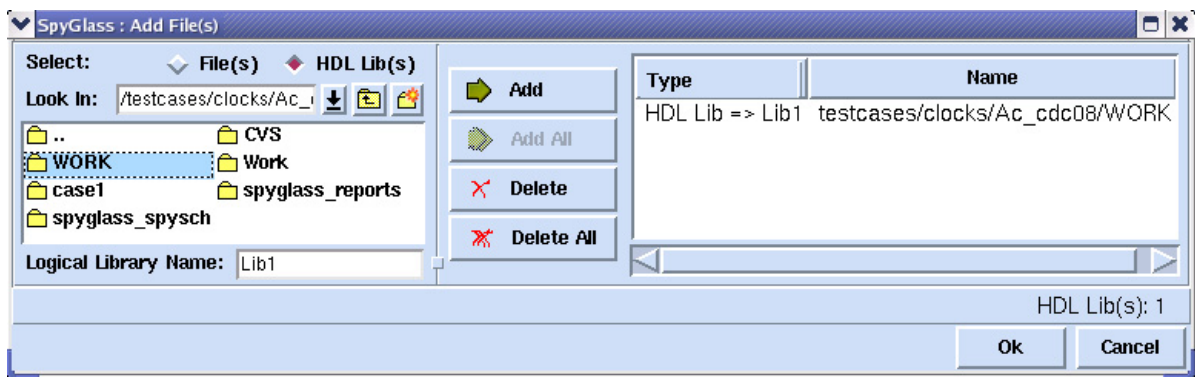


FIGURE 2. Add File(s)

6. Click the *OK* button.

The *HDL Libraries* section now appears, as shown in the following figure:

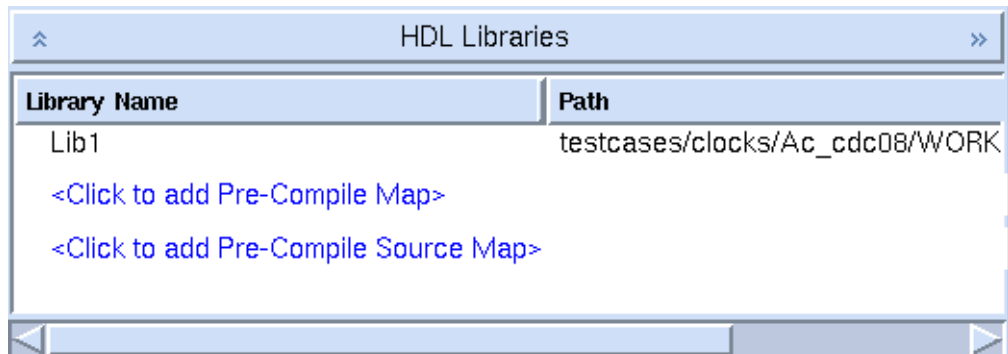


FIGURE 3. HDL Libraries

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. Click the *Click to add Pre-Compile Map* link in the *HDL Libraries* section. The *Precompile File Mapping* dialog appears.

NOTE: To include source files in a library, click the *Click to add Pre-Compile Source Map* link. For details, see [Including Source Files in a Library](#).

- In the *Precompile File Mapping* dialog, click the *Add* button.
A new row appears in this dialog, as shown in the following figure:

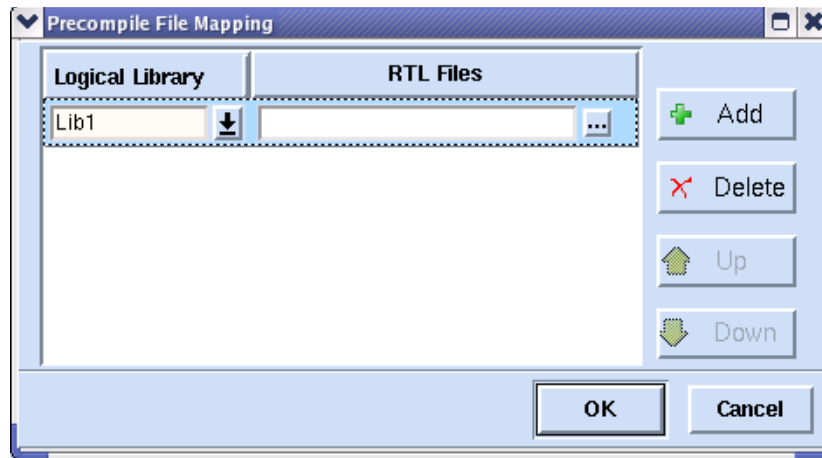



FIGURE 4. Precompile File Mapping

- Click  to select the required logical library name in the *Logical Library* column.

NOTE: You can only select the library name from the list of available library names. Therefore, it is recommended to define the library mapping and then precompile file mapping.

- Click  to select the RTL file to be included in the library.

Repeat this step to specify multiple RTL files.

You can specify wildcard and regular expressions that are automatically expanded when the selection is changed.

- Click the *OK* button.

The HDL files are now included in the library.

This task is equivalent to specifying the following command in the project file:

```
set_option libhdlfiles <library-name> {space-separated file list}
```

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.

Including Source Files in a Library

To add a source file list in a library, click the *Click to add precompile Source Map* link in the *HDL Libraries* section. This displays the *Precompile Filelist Mapping* dialog. The rest of the steps are similar to adding HDL files. For details, see [Including HDL Files in the Logical Library](#).

Alternatively, you can add source files in a library by specifying the following command in the project file:

```
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.

If the precompiled libraries contain errors, the following dialog appears:

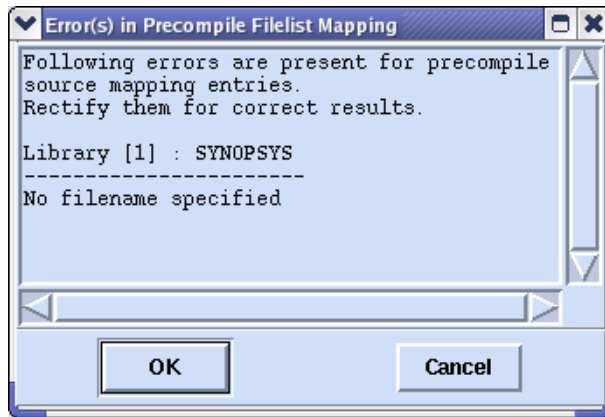


FIGURE 5. Error(s) in Precompile Filelist Mapping

In the above dialog, click the *Cancel* button and go to the *Precompile Map* dialog to correct the directory. See [Debugging Issues in Gate Libraries](#).

Alternatively, you can generate a precompiled library through a project file by specifying the following command while invoking Atrenta Console:

```
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_AUTO_COMPILE` 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.sflib 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, Atrenta Console 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, Atrenta Console 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 Atrenta Console 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.

Atrenta Console performs this check only when the specified libraries contain duplicate cells across different libraries.

NOTE: *Atrenta Console 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
- SYNOPSIS

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 Atrenta Console 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, Atrenta Console 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 }
```

Atrenta Console first checks the current directory for the specified libraries. If the specified libraries are not present in the current directory, Atrenta Console searches in the specified path.

Like all standard Verilog EDA tools, Atrenta Console 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, Atrenta Console 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, Atrenta Console 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. Atrenta Console 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:

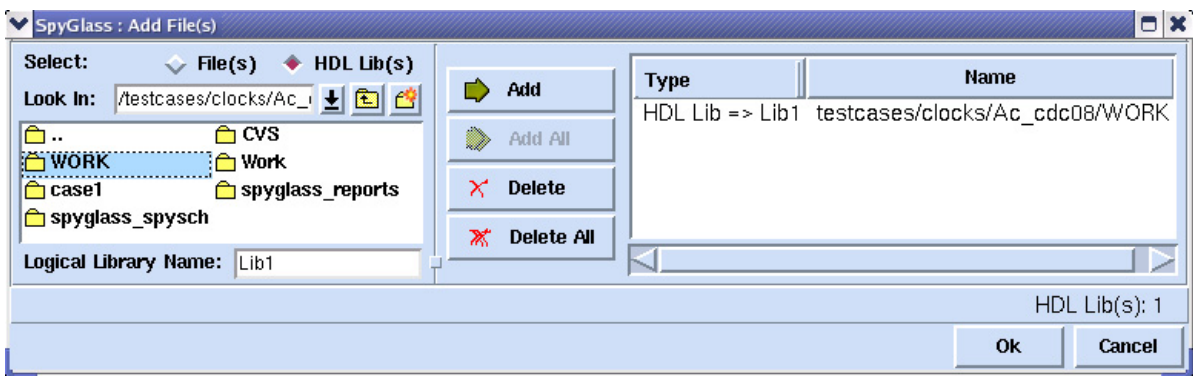


FIGURE 6. Specifying Precompiled Libraries

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 hdlldu 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.
Atrenta Console 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

Atrenta Console 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, Atrenta Console 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, Atrenta Console 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 `hdlldu` 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:

<p>Scenario 1</p> <p>The WRN_405 violations are not reported during the usage of the precompiled dump as the <code>dump_precompile_builtin</code> option is set to no during the creation of that precompiled dump.</p>	<p>Creating a precompiled dump:</p> <pre>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</pre> <hr/> <p>Using the precompiled dump:</p> <pre>set_option top top set_option lib L1 P1 set_option hdlldu yes</pre>
<p>Scenario 2</p> <p>The WRN_405 violations are reported during the usage of the precompiled dump as the <code>dump_precompile_builtin</code> option is set to yes during the creation of the precompiled dump, and the <code>hdlldu</code> option is specified during the usage of the precompiled dump.</p>	<p>Creating a precompiled dump:</p> <pre>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</pre> <hr/> <p>Using the precompiled dump:</p> <pre>set_option top top set_option hdlldu yes set_option lib L1 P1</pre>

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 hdlldu 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 hdlldu 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 hdlldu 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 vhd187
LIBHDL_EXTMAP = .vh93 vhd193
LIBHDL_EXTMAP = .vhd vhd193
```

- 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, refer to the `libhdl_extmap` section in the *Atrenta Console Reference Guide*.

Example

Say you want to compile the following source files in the same SpyGlass run:

<u>L1.f</u>	<u>L2.f</u>	<u>L3.f</u>
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 vhd187
LIBHDL_EXTMAP = .vh93 vhd193
```

```
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
```

Atrenta Console 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.

Viewing Compilation Language for Different File Extensions

To view compilation language for various file extensions set in the .spyglass.setup file, perform the following steps:

1. Click the *Design Setup* tab.
2. Click the *Add Design Files* tab.
3. Click the *More Actions* option. A list appears.
4. Select the *Show HDL Library files Extension Mapping* option from the list.

The *HDL Library files Extension Mapping* dialog appears that shows the extension mapping set by using the LIBHDL_EXTMAP variable in .spyglass.setup.

The following figure shows the *HDL Library files Extension Mapping* dialog:

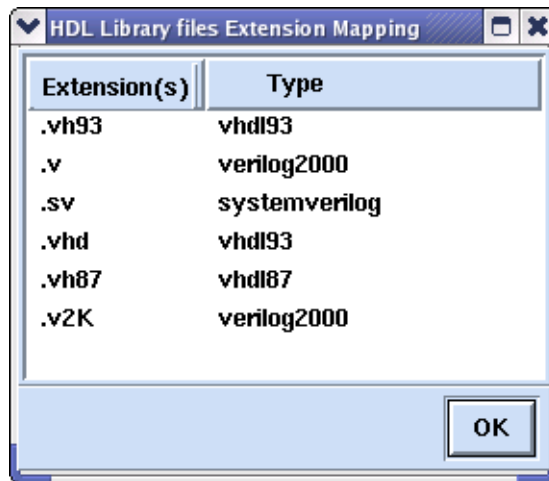


FIGURE 7. HDL Library File Extension Mapping

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:

<code>-hdlin_translate_off_skip_text</code>	<code>-pragma</code>	<code>-top</code>
<code>-allow_module_override</code>	<code>-nodefparam</code>	<code>-sort</code>
<code>-allow_celldefine_as_top</code>	<code>+define</code>	<code>+resetall</code>
<code>-enable_hdl_encryption</code>	<code>+incdir</code>	<code>-param</code>
<code>-hdlin_synthesis_off_skip_text</code>	<code>-sfcu</code>	<code>-macro_synthesis_off</code>
<code>-relax_hdl_parsing</code>		

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:

<u>L1.f</u>	<u>L2.f</u>	<u>L3.f</u>
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

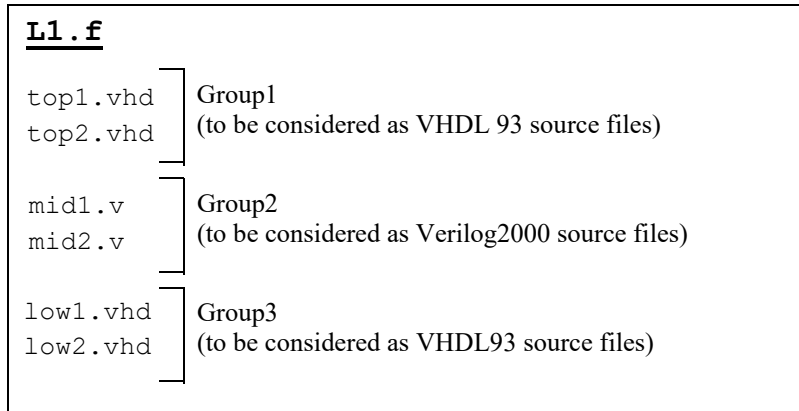
Modules of different source file types may have dependencies among them. Therefore, you should define them in the order of their dependencies in the file specified by the `libhdlf` command.

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:



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 console 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:

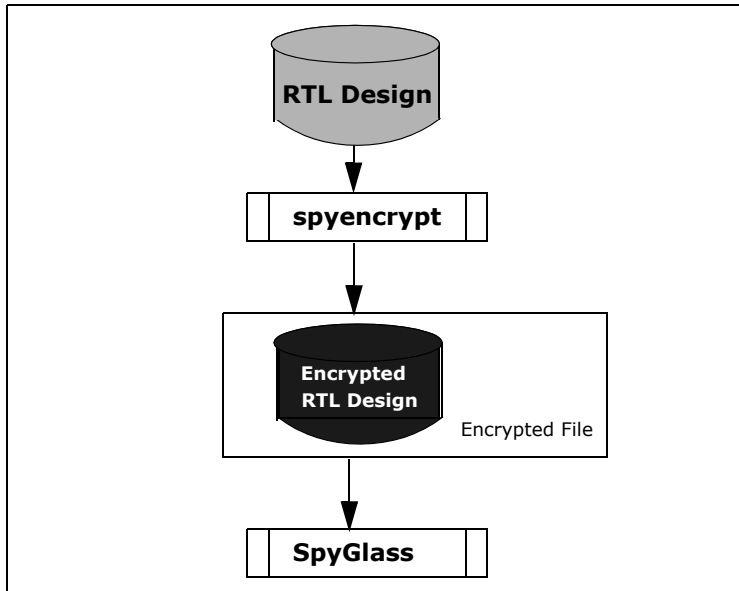


FIGURE 8. IP Encryption Use Model in SpyGlass

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:

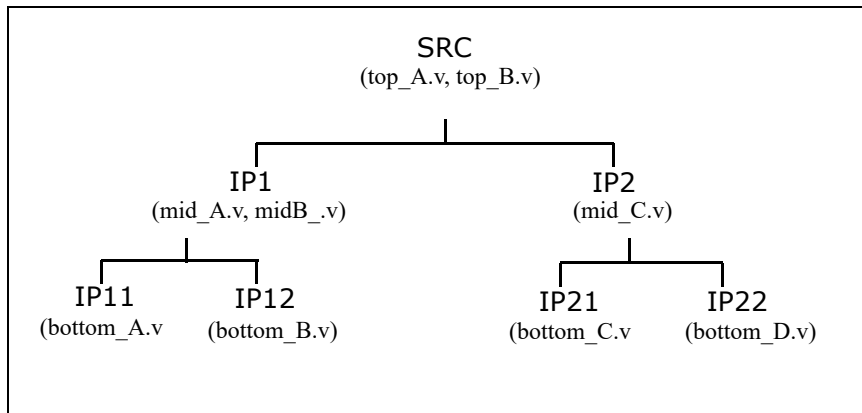


FIGURE 9. Directory Structure - Design Files to be Encrypted

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 10](#).

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:

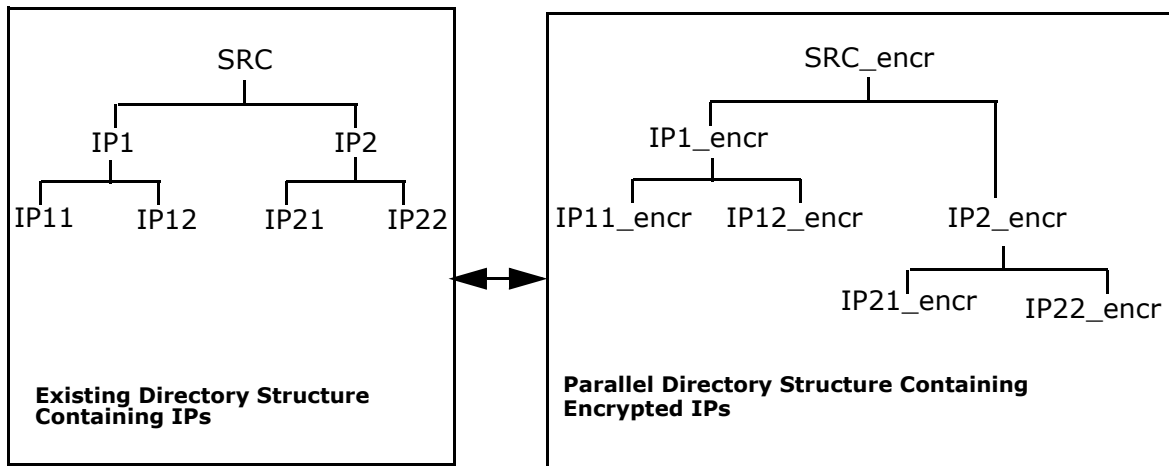


FIGURE 10. Directory Structure - Encrypted Design Files

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
<code>enc/top.v.sge</code>	N.A	FAIL	Encrypted File can't be encrypted
<code>src/mid.v.gz</code>	N.A	FAIL	GZ File can't be encrypted
<code>src/bottom.v</code>	<code>enc/bottom.v.sge</code>	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:

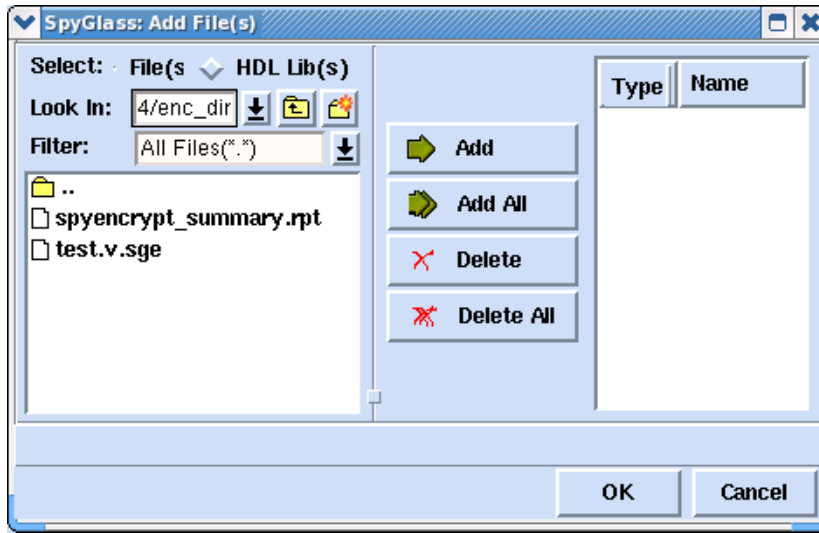


FIGURE 11. Specifying Encrypted Files

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:

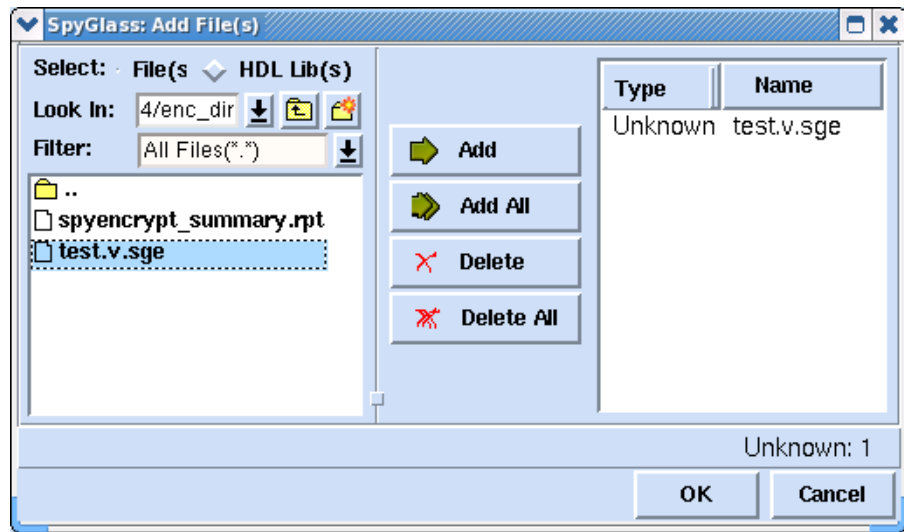


FIGURE 12. Specifying Encrypted Files - Add File(s)

5. In the above dialog, click on the *Unknown* text under the *Type* column. A drop-down list appears, as shown below:

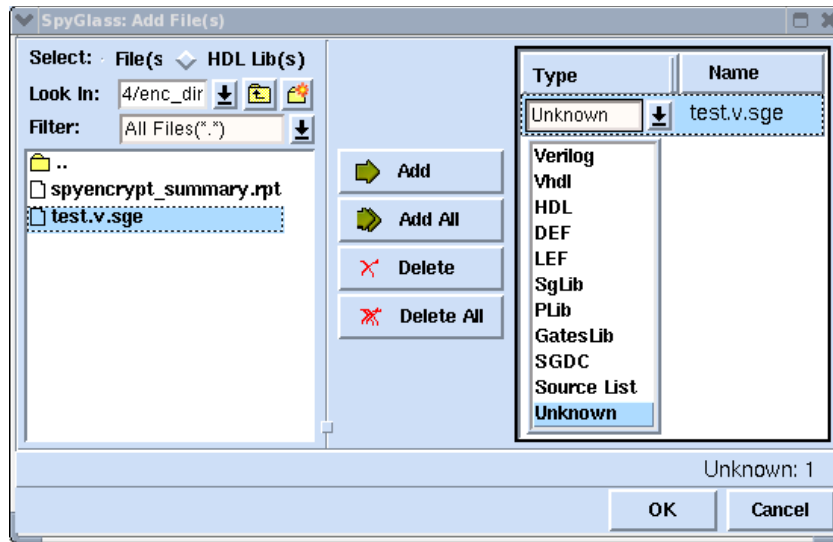


FIGURE 13. SpyGlass Add File(s)

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
<code>\myLibrary.myEntity(myArch)</code>	Architecture myArch of entity myEntity from logical library myLibrary
<code>\myEntity(myArch)</code>	Architecture myArch of entity myEntity from logical library work
<code>\myLibrary.myEntity</code>	MRA Architecture of entity myEntity from logical library myLibrary
<code>\myLibrary.myConfigDecl</code>	Configuration declaration myConfigDecl from logical library myLibrary
<code>myName</code>	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, Atrenta Console 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 1](#), SpyGlass requires the DesignCompiler license to generate netlist for these modules.

[Table 2](#) 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, Atrenta Console 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 1 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_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	DW_fp_div

TABLE 1 DesignWare Modules that Require Design Compiler License

DW_fpflt2i	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_fifocntl_2c_df	DW_log2
DW_exp2	DW_fp_div_seq	DW_thermdec
DW_data_qsync_lh	DW_dct_2d	DW_fifo_2c_df
DW_asymfifocntl_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	DW_sincos

TABLE 1 DesignWare Modules that Require Design Compiler License

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 2 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
DW_div	DW_gray2bin	DW_inc_gray

TABLE 2 DesignWare Modules not Requiring Design Compiler License

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. Atrenta Console supports different types of pragmas for Verilog and VHDL code.

Supported Pragmas for Verilog

Atrenta Console supports `translate_off/translate_on` pragmas for Verilog. Within these pragmas, Atrenta Console ignores the code for compilation, syntax checks, etc.

Supported Pragmas for VHDL

Atrenta Console 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™ 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. Atrenta Console processes only those modules for black box inference that are not defined in any of these files.

Atrenta Console 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 Atrenta Console 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 Atrenta Console on a machine configured with more physical RAM and swap space. By default, Atrenta Console 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 Atrenta Console 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, refer to the Memory Reduction Feature section of Atrenta Console Reference Guide.

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, refer to *Atrenta Console Reference Guide*.

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 Atrenta Console 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/aux/target_libs/generic/rtl.c.prim.v` and `<your-inst-dir>/SpyGlass-x.y.z/SPYGLASS_HOME/aux/target_libs/generic/rtl.c.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*.

For more details on *disable_hdlin_translate_off_skip_text command*, refer to *disable_hdlin_translate_off_skip_text* section in the Atrenta Console Reference Guide.

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, Atrenta Console 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.

If you do not set a top-level module during the *Design Setup* stage and directly proceed to the next stage, Atrenta Console performs appropriate actions as discussed below:

- If you proceed to the next stage after running the design read process, and if there are multiple top-level modules in your design but none of the modules are set as a top-level module, the following *Warning* dialog appears:

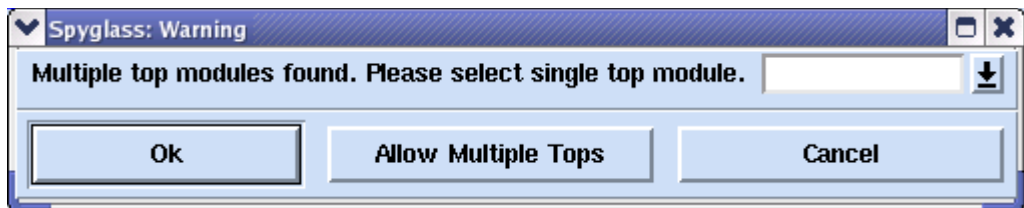



FIGURE 14. Warning: Multiple Top Modules

In the above dialog, click  and select the required module from the drop-down list.

If you want to set all the modules displayed in that list as top-level modules, click the *Allow Multiple Tops* button.

- If you proceed directly to the next stage without running the design read process, the following *Warning* dialog appears:

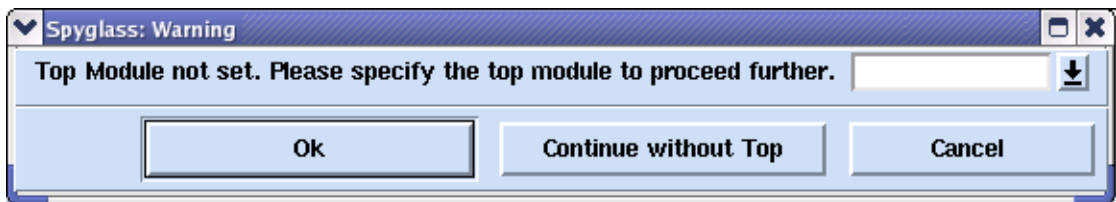


FIGURE 15. Warning: Top Module Not Set

In the above dialog, specify the name of the top-level module.

If you want to run SpyGlass analysis without any top-level module, click the *Continue Without Top* button.

- If your design has a single top-level module that you have not set as a top-level module during the *Design Setup* stage, Atrenta Console automatically infers that module as a top-level module based on the structure of the design. In this case, no *Warning* dialog appears.

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.

Atrenta Console 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. Atrenta Console 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

Atrenta Console 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:

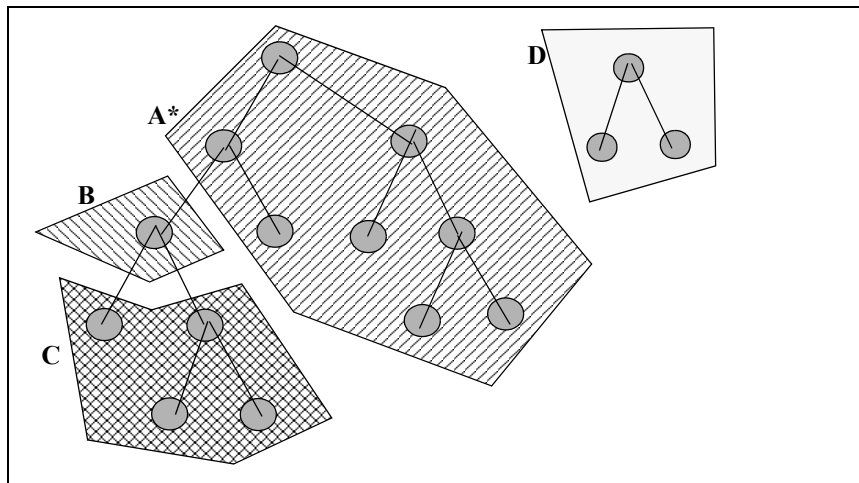


FIGURE 16. Design Hierarchy for Use of Top-Level and Stop Features

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 Atrienta Console 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 3 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  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
<code>set_option ignorefile {a*}</code>	Ignores all files (in the current directory) whose names match the wildcard <code>a*</code> expression. For example, <code>a1</code> , <code>aa1</code> , and <code>abc</code> .
<code>set_option ignorefile {dir1/*}</code>	Ignores all files in the <code>dir1</code> directory
<code>set_option ignorefile {dir?/*}</code>	Ignores all files in directories that match the wildcard expression <code>dir?</code> . For example, <code>dir1</code> , <code>dir2</code> , and <code>dir3</code> .

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 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.

- Right-click on a design unit or a module in the *Module View* page and select the *Set Module Ignore* option from the shortcut menu.

The *Set Module Ignore* dialog appears showing the name of the selected design unit or module:

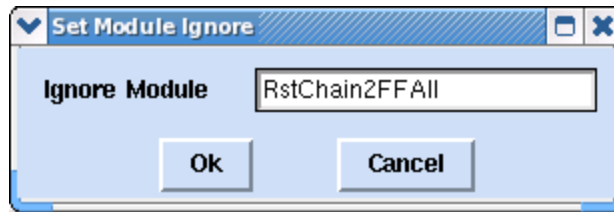



FIGURE 17. Set Module Ignore

- Click the *OK* button in the above dialog.

After performing the above steps, the selected design unit or module is ignored and the  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

Format	Description
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
<code>set_option ignoredu {lib1.*}</code>	Ignores all design units picked from the <code>lib1</code> logical library
<code>set_option ignoredu {e.*}</code>	Ignores all architectures of the <code>e</code> entity
<code>set_option ignoredu {*.e}</code>	Ignores all entities named <code>e</code> 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 <code>check_celldefine</code> 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

Overview

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:

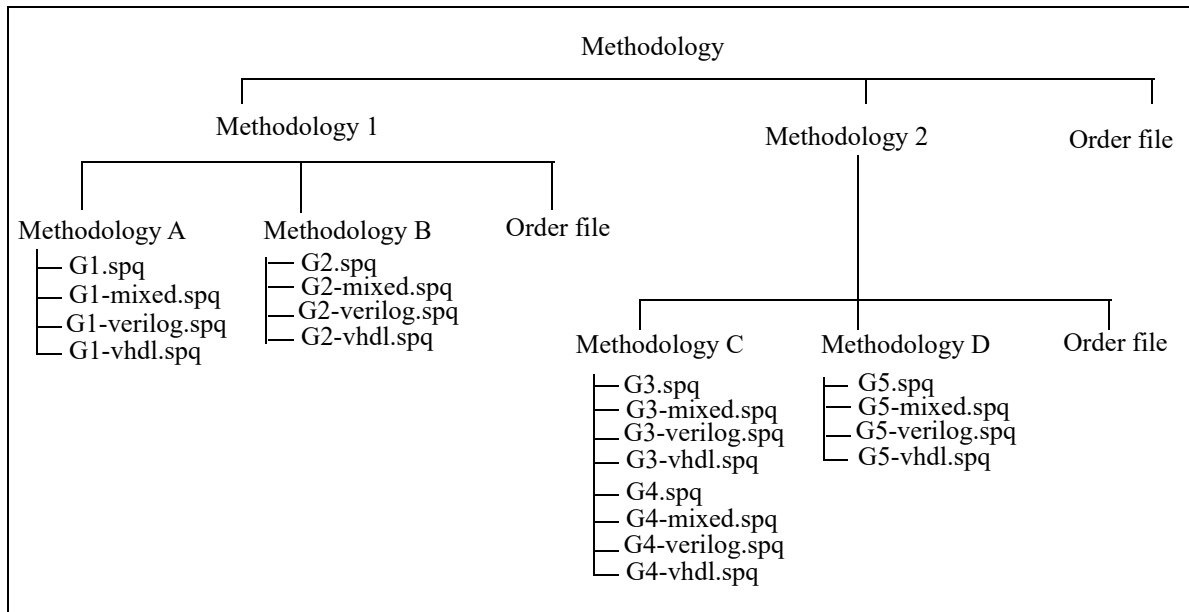


FIGURE 1. Sample Methodology Structure

NOTE: Each methodology should contain an *Order File* that contains entries of goal files paths related to a methodology directory.

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 goal 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 Atrenta Console.

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:

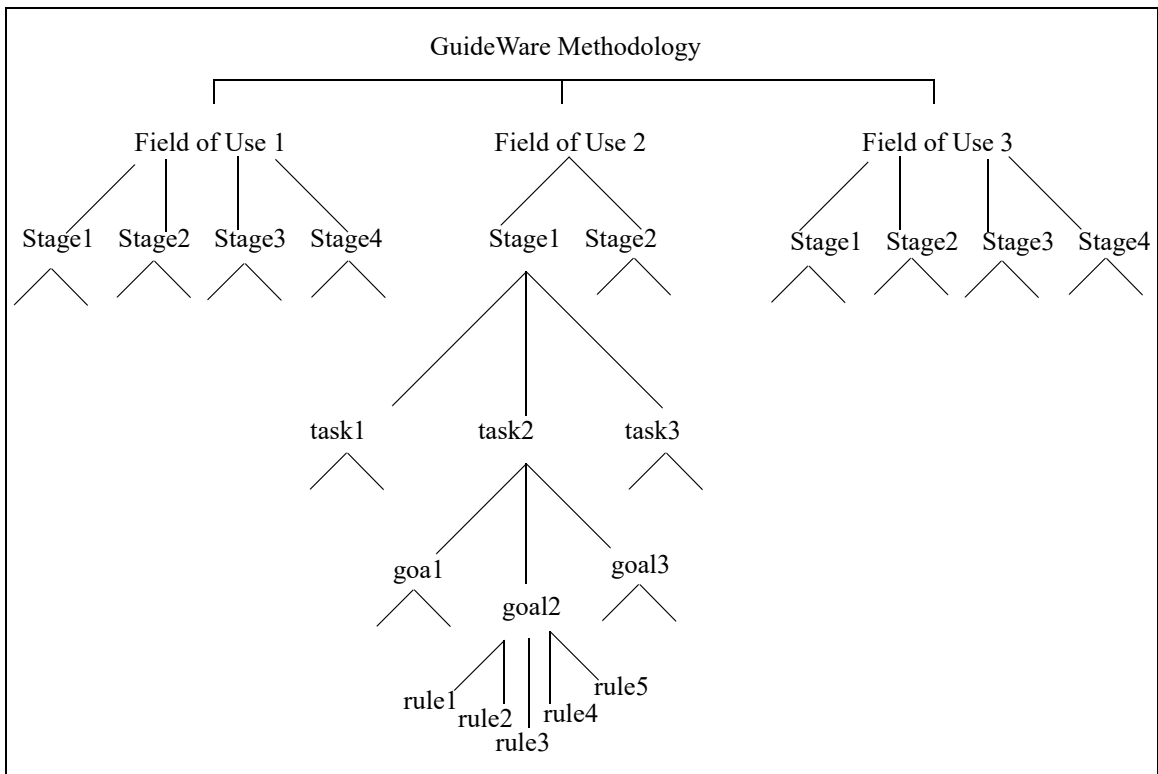


FIGURE 2. GuideWare Methodology Structure

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 <i>SpyGlass GuideWare User Guide</i> 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:

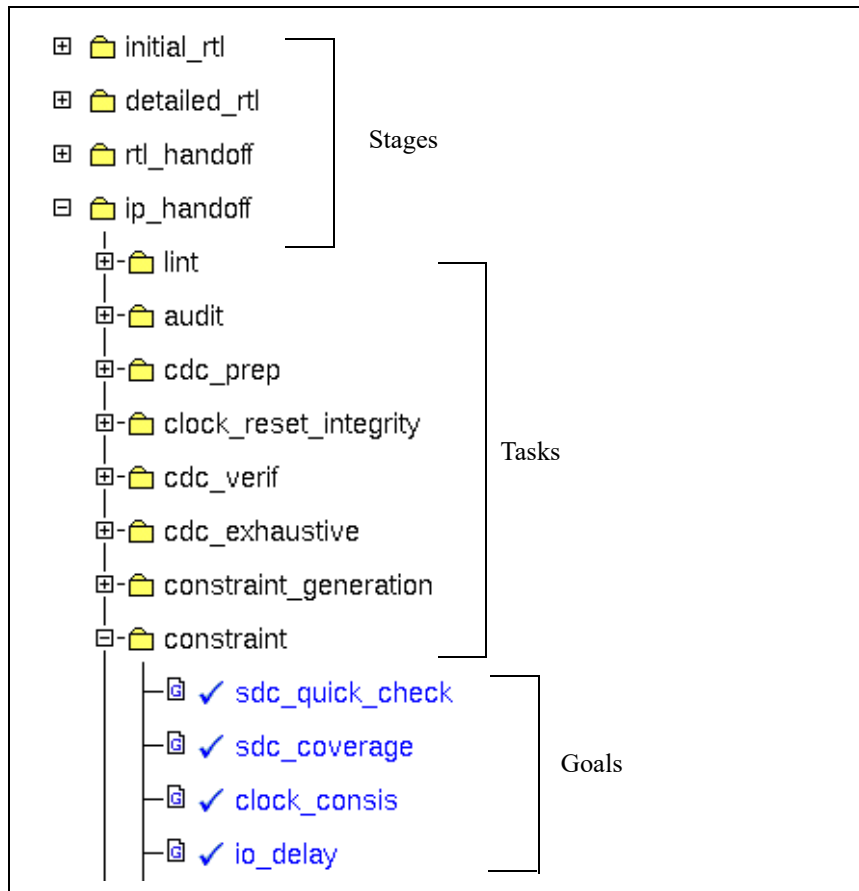


FIGURE 3. Goal Alignment in New_RTL Field of Use

Specifying an Active Methodology

An active methodology refers to a default methodology that gets loaded when you start Atrenta Console.

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:

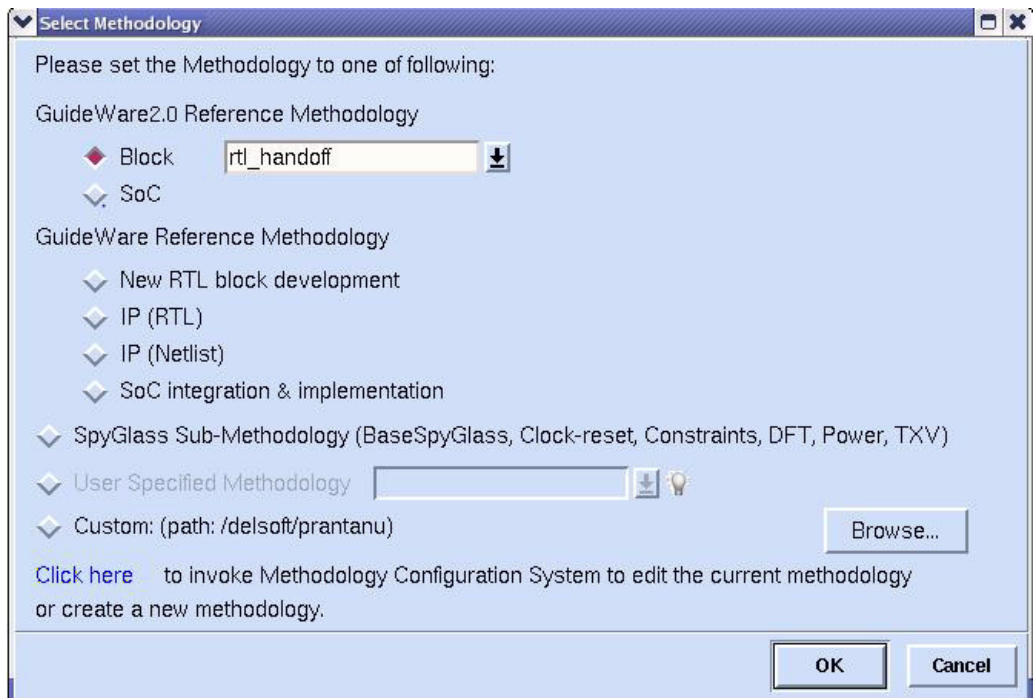



FIGURE 4. Select Methodology

Specifying an Active Methodology

In the above dialog, you can select any option described in the following table:

Option	Description
New RTL block development	(Default) Select this option to load GuideWare goals for the New_RTL methodology, installed at: <your-inst-dir>/SPYGLASS_HOME/GuideWare/New_RTL directory
IP (RTL)	Select this option to load GuideWare goals for the IP_RTL methodology, installed at: <your-inst-dir>/SPYGLASS_HOME/GuideWare/IP_RTL directory
IP (Netlist)	Select this option to load GuideWare goals for the IP_Netlist methodology, installed at: <your-inst-dir>/SPYGLASS_HOME/GuideWare/IP_Netlist directory
Soc Integration & implementation	Select this option to load GuideWare goals for the SoC methodology, installed at: <your-inst-dir>/SPYGLASS_HOME/GuideWare/SoC directory
Guideware 2.0 Reference Methodology	Select one of the following options to use Guideware 2.0: <ul style="list-style-type: none"> ● Block ● SoC
SpyGlass Sub-Methodology	Select this option to load SpyGlass goals, installed at: <your-inst-directory>/SPYGLASS_HOME/Methodology directory

User Specified Methodology	<p>Select this option to specify your own custom methodology.</p> <p>After selecting this option, click the  button and select a custom methodology from the drop-down list.</p> <p>Values in this drop-down list are picked from the methodologies specified in the <code>METHODOLOGY_SEARCH_PATH</code> variable in the <code>.spyglass.setup</code> file.</p> <p>The advantage of using this option over the <i>Custom</i> option is that you do not need to browse to the path of a custom methodology every time you open the <i>Select Methodology</i> dialog.</p>
Custom	<p>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.</p> <p>After selecting this option, click the <i>Browse</i> button to browse to the required directory.</p>

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, Atrenta Console 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, Atrenta Console 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, Atrenta Console 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, Atrenta Console 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:

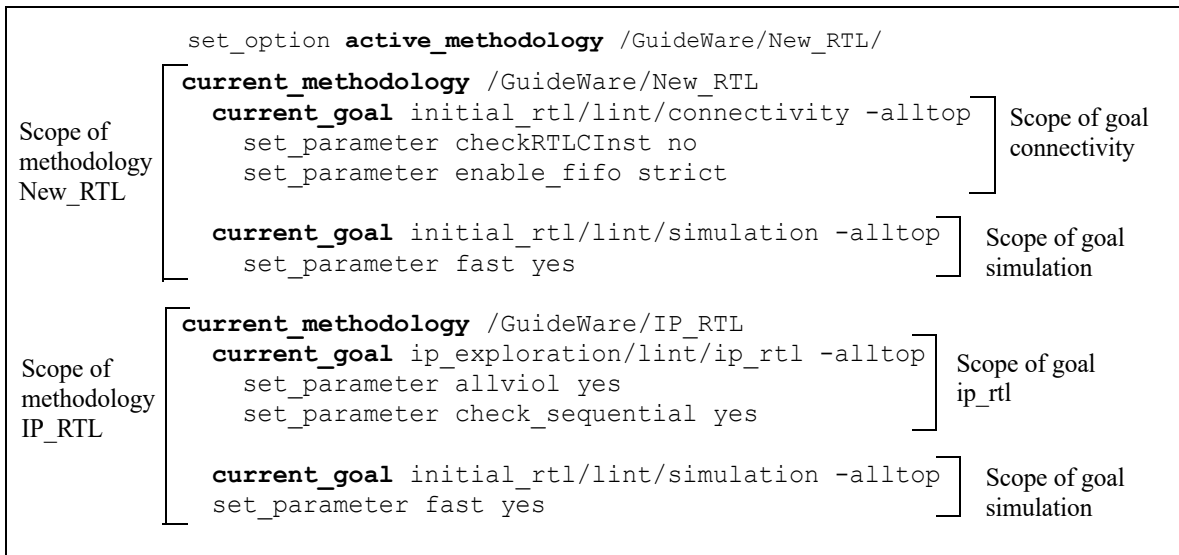


FIGURE 5. Define Methodology Scope

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

Specifying a Current Methodology

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, Atrenta Console 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:

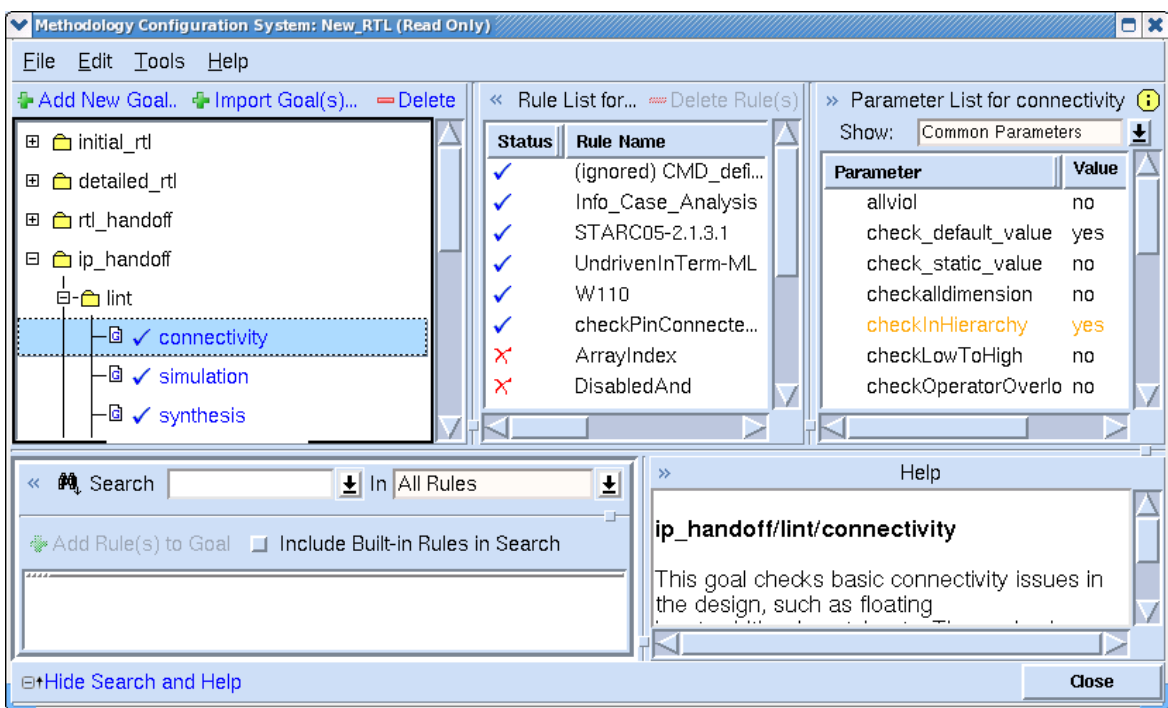


FIGURE 6. Methodology Configuration System

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, Atrenta Console 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, refer to <i>The Methodology Configuration System Menu Bar</i> section of <i>Atrenta Console Reference Guide</i> .
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 <i>Rules List</i> section, and select the <i>Configure Columns</i> shortcut menu option. Refer to the Configuring Columns for details on how to show/hide a rule.
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:

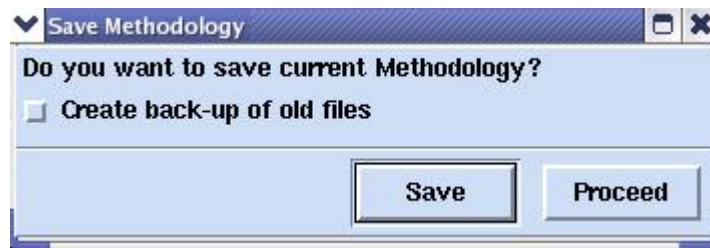
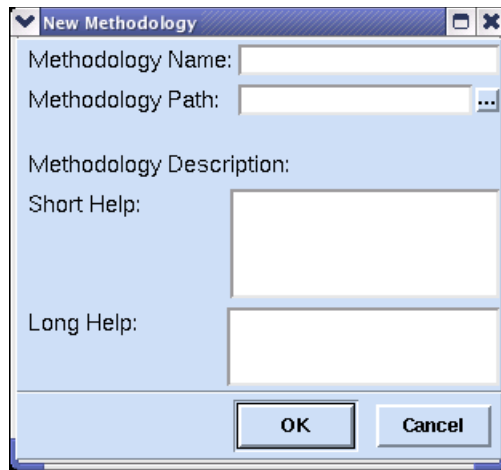


FIGURE 7. Save Methodology

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:

Configuring a Methodology

**FIGURE 8.**

- 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 <i>Goal Selection</i> section
Long Help	Specifies a detailed description of the new methodology being created. This description is saved in the order file.

- 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:

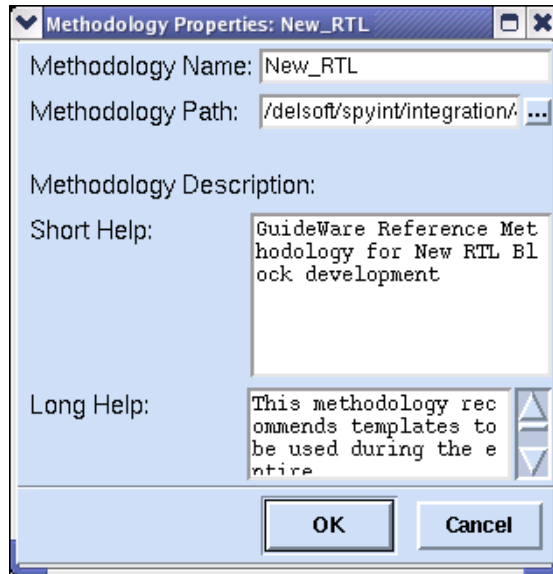


FIGURE 9. Methodology Properties

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, Atrenta Console 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:

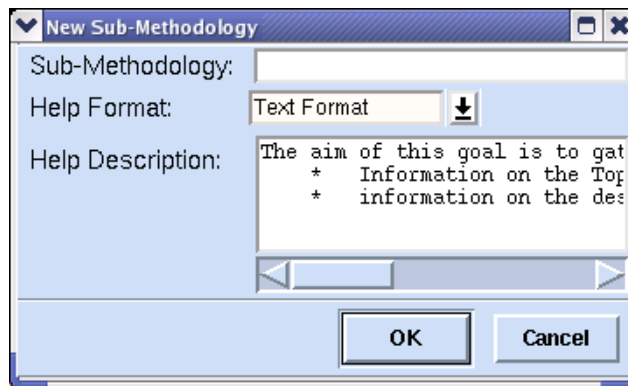


FIGURE 10. New Sub Methodology - Text Format

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:

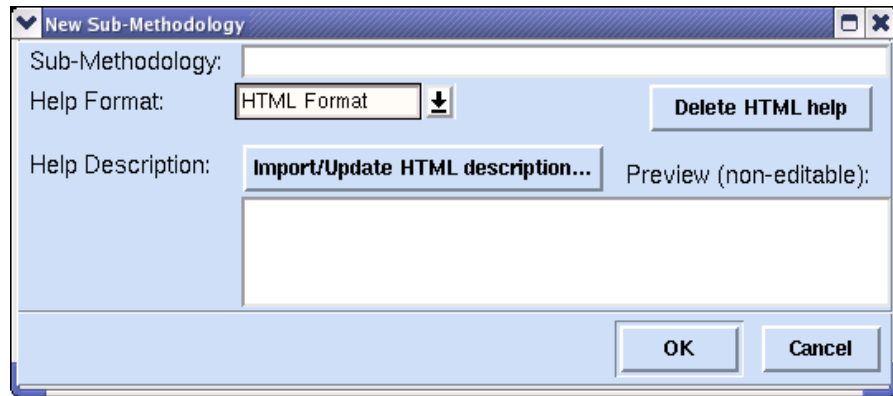


FIGURE 11. New Sub Methodology - HTML Format

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, ``).

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:

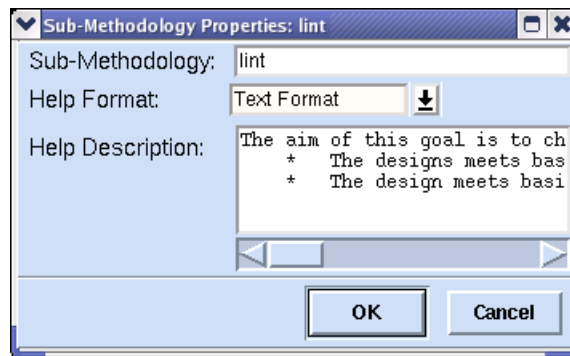


FIGURE 12. Sub-Methodology Properties

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:

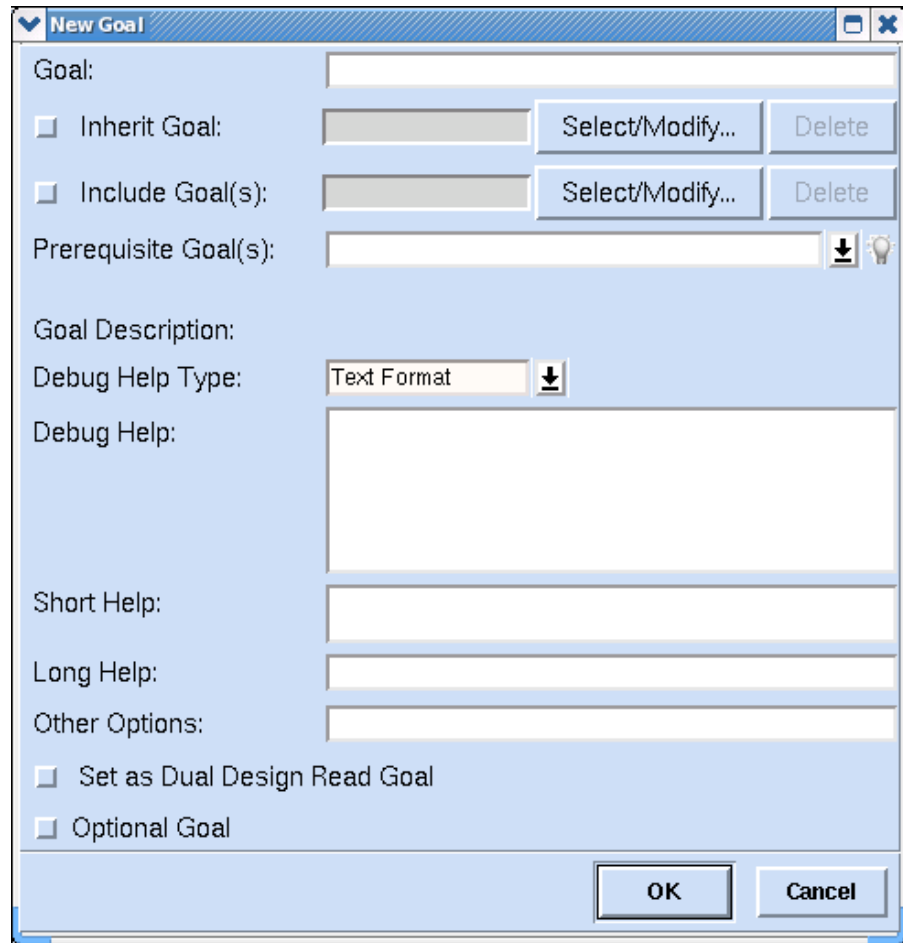


FIGURE 13. New Goal

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 <i>Select/Modify</i> button. The <i>Inherit Goal</i> 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 <i>Delete</i> button.

Field	Description
Include Goal(s)	<p>Specifies a goal to be included in the new goal being created. For details on included goal, see Including and Inheriting GuideWare Goals.</p> <p>To include a goal, select the <i>Select/Modify</i> button. The <i>Include Goal(s)</i> dialog appears in which you can select the required goals.</p> <p>Select or deselect the corresponding checkbox to enable or disable the included goals.</p> <p>To delete the included goals from the goal being created, click the <i>Delete</i> button.</p>
Prerequisite Goals	<p>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 <i>Prerequisite Goals</i> field, and select the required goal(s) from this list.</p> <p>The prerequisite goals appear in the <i>Prereq Goals</i> column of the goal selection window (see Selecting a Goal).</p>
Debug Help Type	<p>Specifies the format (text or HTML) in which the goal debug help should be visible.</p>
Debug Help	<p>Specifies the debug information that helps you debug issues reported by this goal.</p> <p>This help is visible during the <i>Analyze Results</i> stage when you select the <i>Goal Debug Help</i> option in the <i>Help</i> section of the <i>Results</i> pane.</p>
Short Help	<p>Specifies the short description of the goal.</p>
Long Help	<p>Specifies the detailed description of the goal.</p> <p>The long help is visible in the <i>Help</i> window under the <i>Methodology Configuration System</i> and the <i>Select Goal</i> tab when modified methodology is loaded.</p>
Other Options	<p>Specifies additional command-line options.</p>

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:

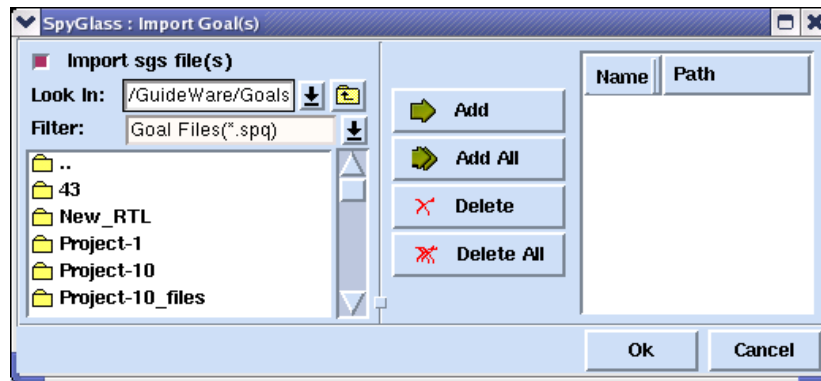


FIGURE 14. Import Goal(s)

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:

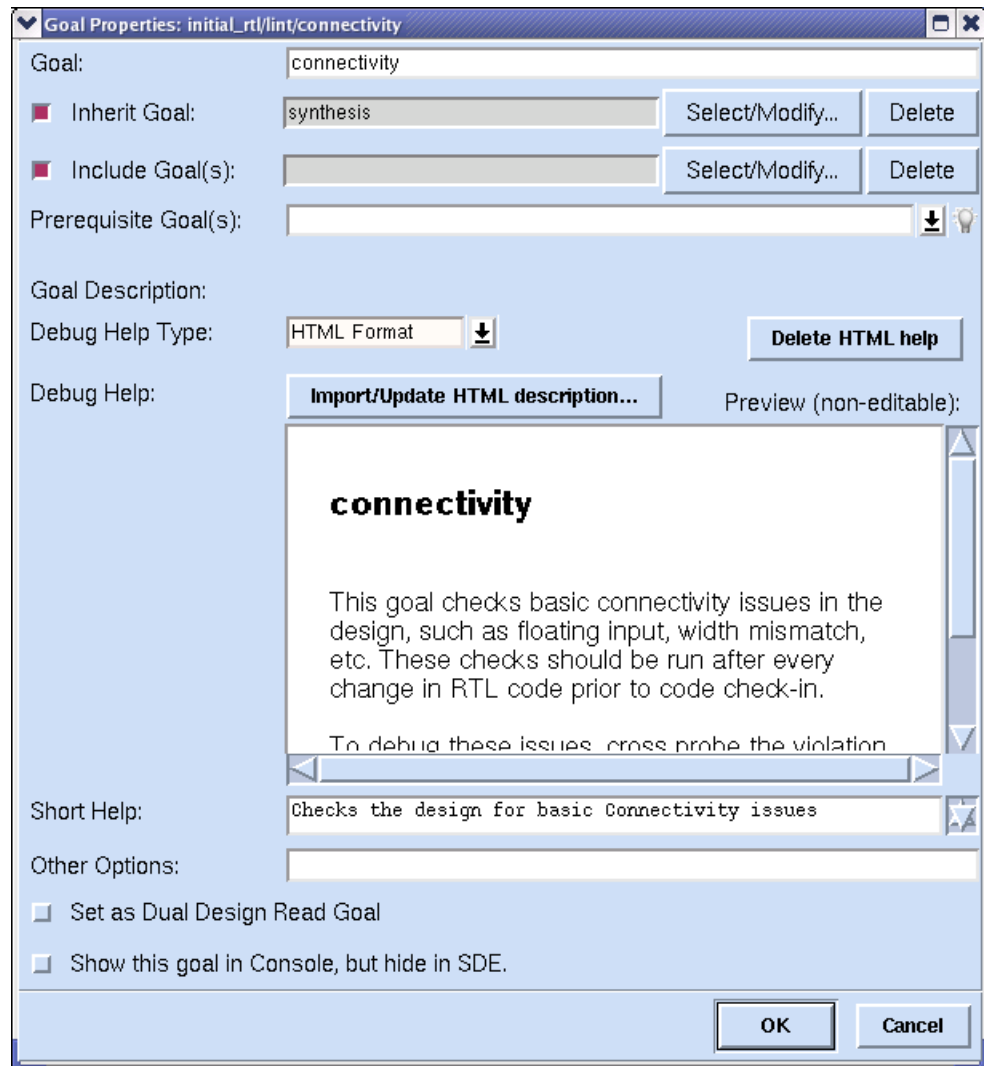


FIGURE 15. Modify Goal Properties

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, Atrenta Console 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:

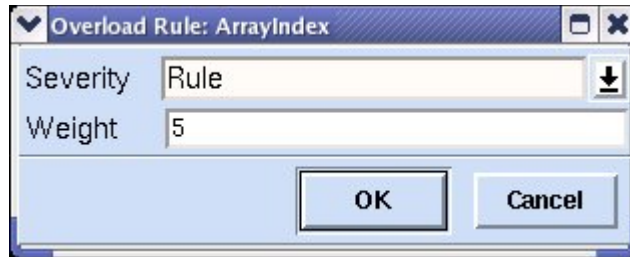


FIGURE 16. Overload Rule

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, Atrenta Console 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:

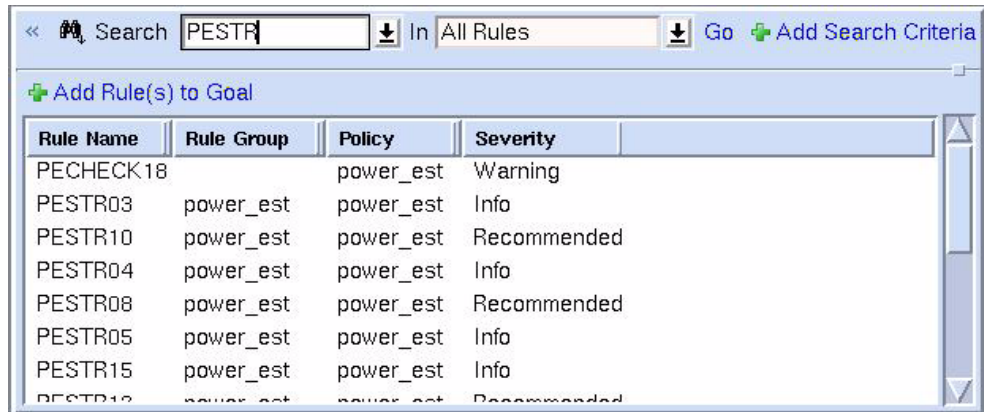


FIGURE 17. Search Rules

In the above section, specify the search text in the *Search* textbox and click the *Go* link. Once you click the *Go* link, Atrenta Console displays all the rules matching the specified search criteria in a spreadsheet format.

By default, Atrenta Console 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, Atrenta Console adds additional fields, as shown in the following figure:

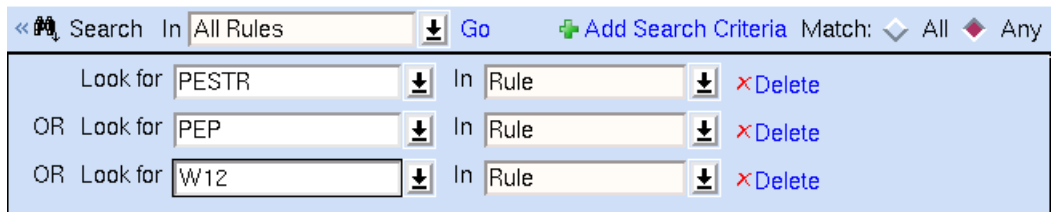


FIGURE 18. Search Fields

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
checkRTLInst	yes
checkunpackdimension	no
disable_rtl_deadcode	no

FIGURE 19. 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 <i>MCS</i> 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:

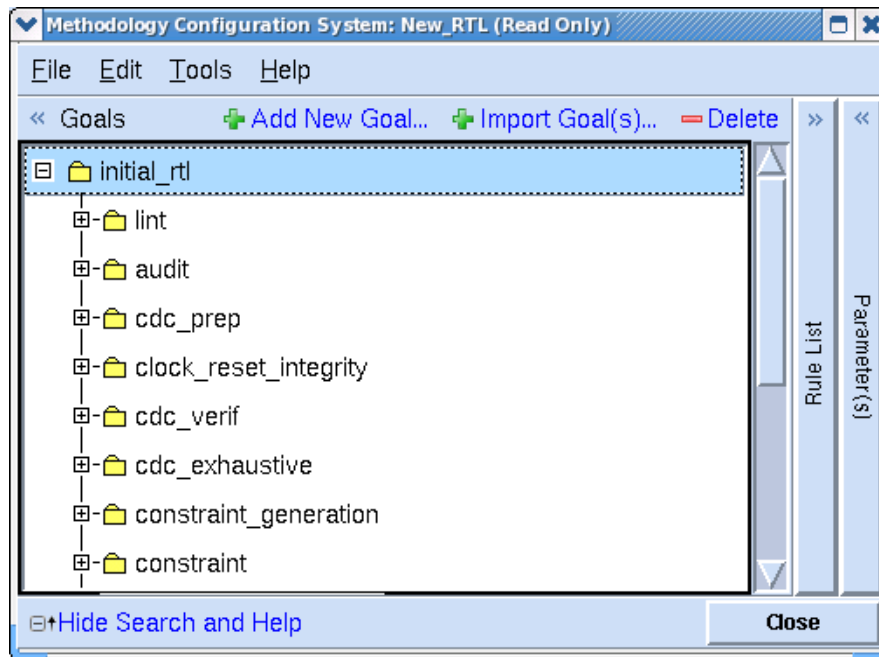


FIGURE 20. MCS Window

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:

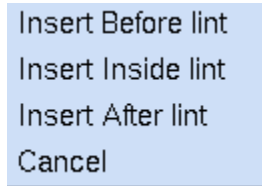


FIGURE 21. Drag and Drop Methodologies - Right-Click Menu

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:

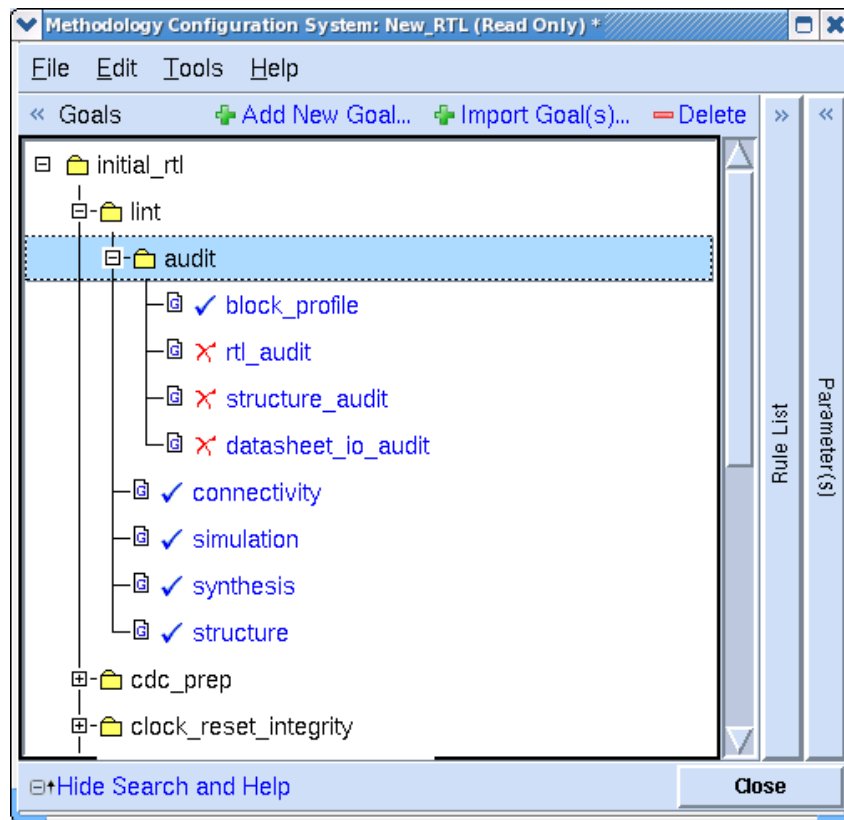


FIGURE 22. MCS Window - Methodology/Goal List

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:

Insert Inside clock_reset_integrity
Cancel

FIGURE 23. Drag and Drop Goals - Right-Click Menu

- 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:

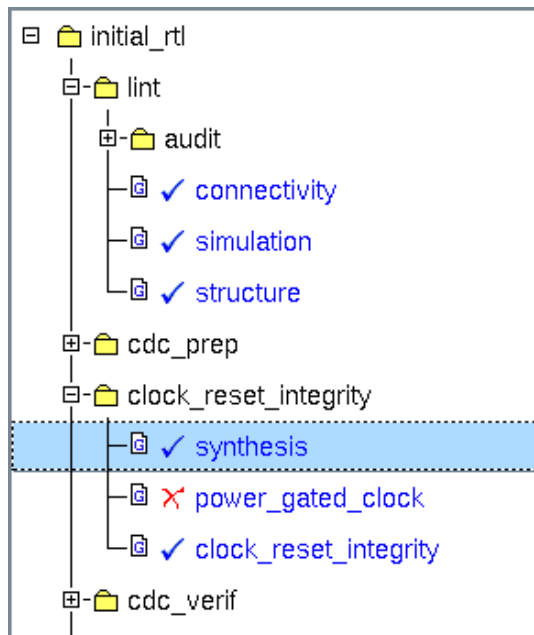
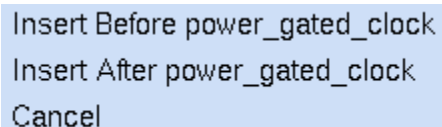


FIGURE 24. MCS Window - Goal List

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:

- Expand the *clock_reset_integrity* sub-methodology.
- 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 25. 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:

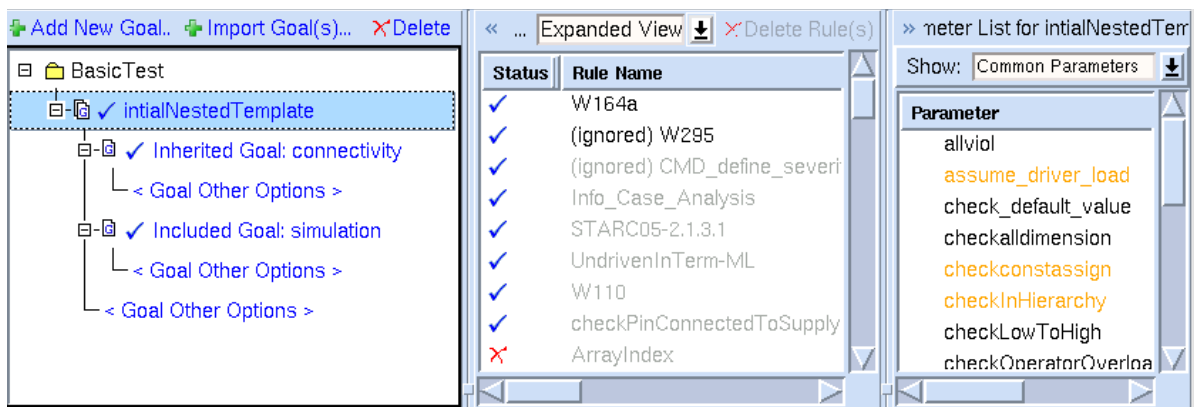


FIGURE 26. Goal Tree

In the above figure, the `initialNestedTemplate` 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, Atrenta Console 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, Atrenta Console 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, Atrenta Console reports the appropriate error message.

Atrenta Console 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_METHODODOLOGY_DIR/a/b/c/
goal1
```

To use the M2 methodology, set the `MY_CORPORATE_METHODODOLOGY_DIR` environment variable to point to the M1 directory before invoking SpyGlass, as shown below:

```
setenv MY_CORPORATE_METHODODOLOGY_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 `METHODODOLOGY_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 $METHODODOLOGY_HOME/a/b/c/goal1
```

The `METHODODOLOGY_HOME` variable is set by SpyGlass automatically when a methodology loaded. Do not define this variable. If it is set, Atrenta Console 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, Atrenta Console overrides and merges the specified overload specifications with subsequent specifications.

If you specify different severity labels in these specifications, Atrenta Console 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

Atrenta Console 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:

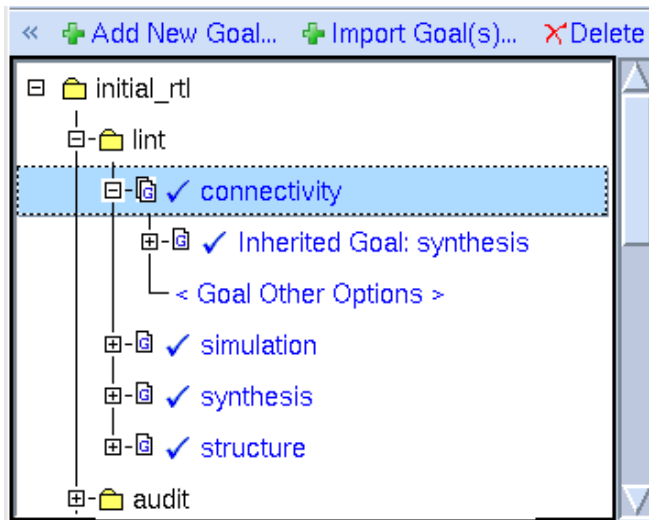


FIGURE 27. Goal Inheritance

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:

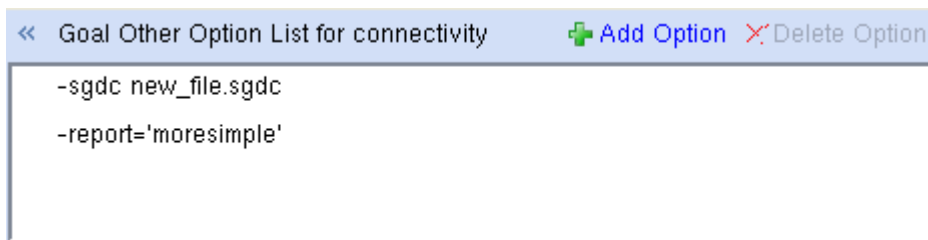


FIGURE 28. Goal Other Option List for Connectivity

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:

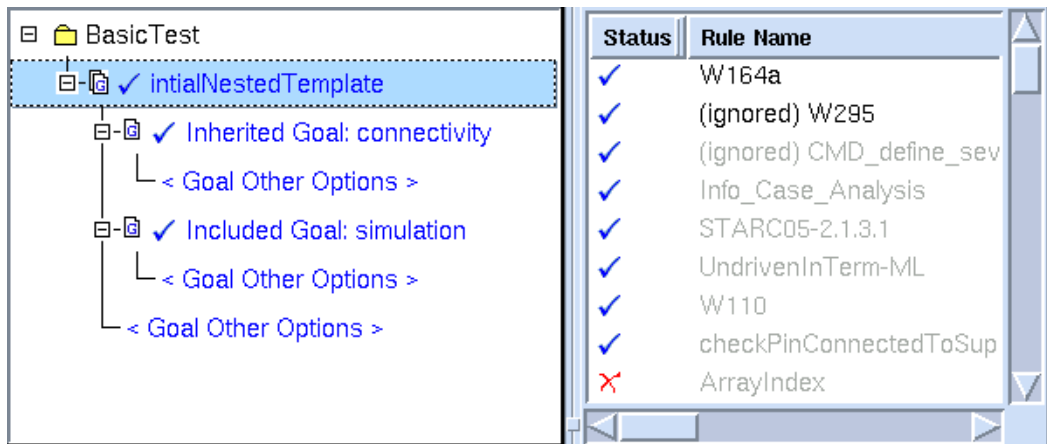


FIGURE 29. Expanded View

- *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 Group
✓	W164a	W164,L
✓	(ignored) W295	Synthes
✓	Inherited Goal: connectivity	
✓	(ignored) CMD_define_severity02	CMD_d
✓	Info_Case_Analysis	INFORM
✓	STARCO5-2.1.3.1	RTL_De
✓	UndrivenInTerm-ML	Undrive
✓	W110	Instanc
✓	checkPinConnectedToSupply	ELECTF

FIGURE 30. 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, Atrenta Console displays goals of the specified custom methodology under the *Select Goal* tab, as shown in the following figure:

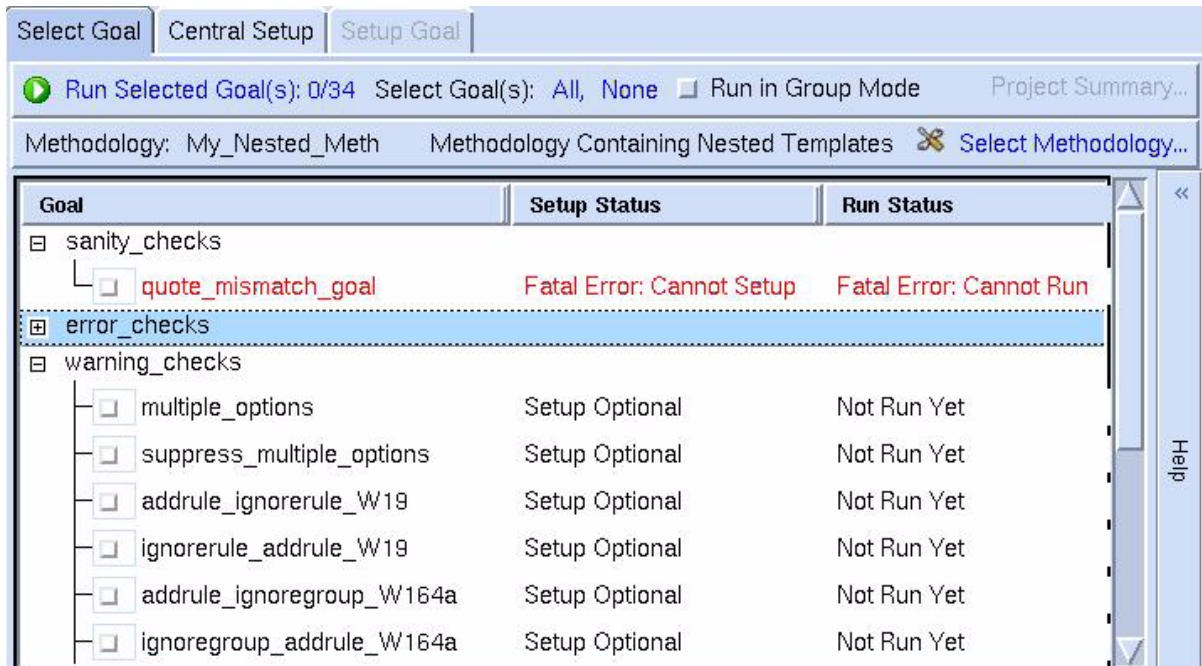
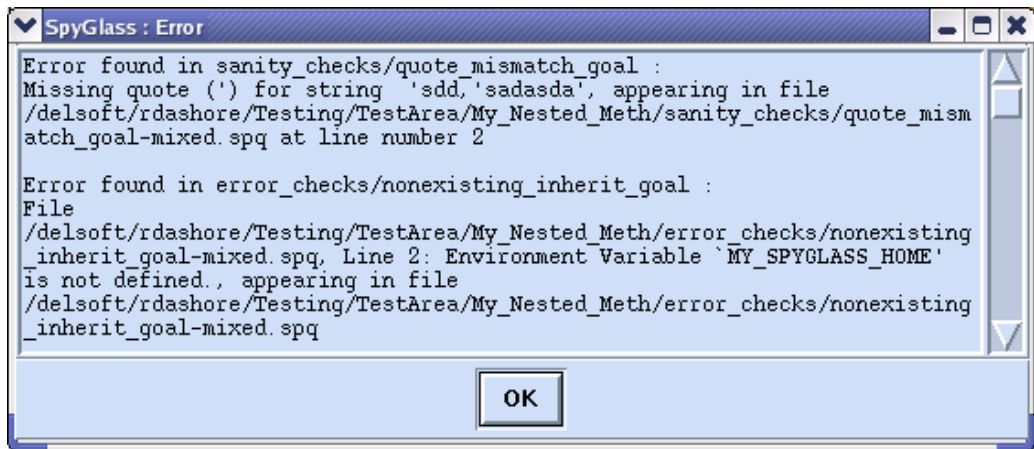


FIGURE 31. Goal List for Custom Methodology

In the above figure, goals highlighted in red indicate some error(s) in such goals. When you open the *Methodology Configuration System* window, Atrenta Console first displays the error details of all such goals in the *Error* dialog, as shown in the following figure:

Selecting a Custom Methodology

**FIGURE 32.** 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:

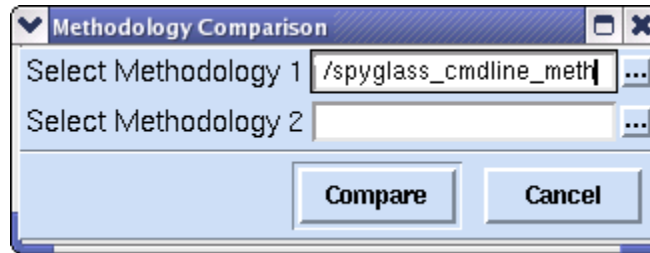


FIGURE 33. Specify Methodologies for Comparison

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:

Comparing Methodologies

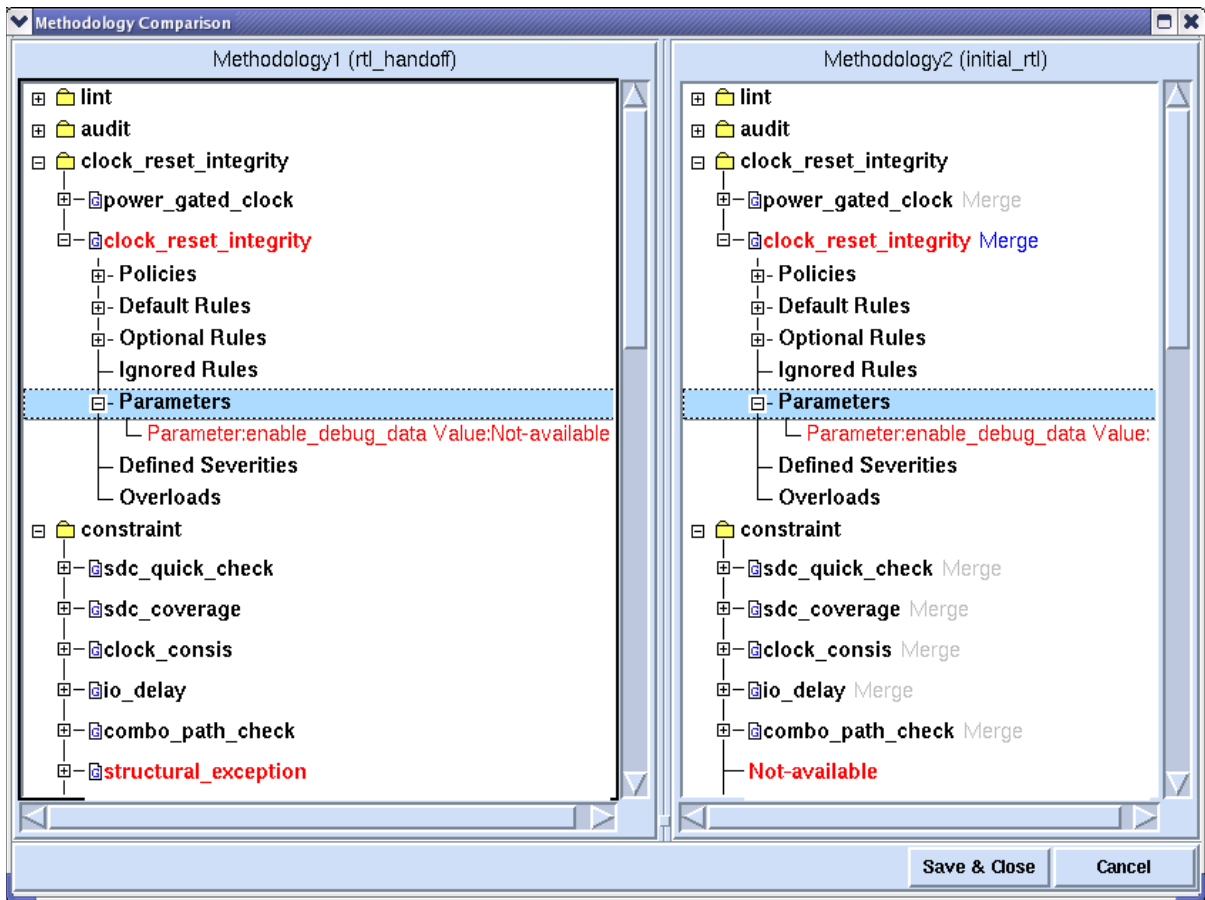


FIGURE 34. Methodology Comparison Results

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:

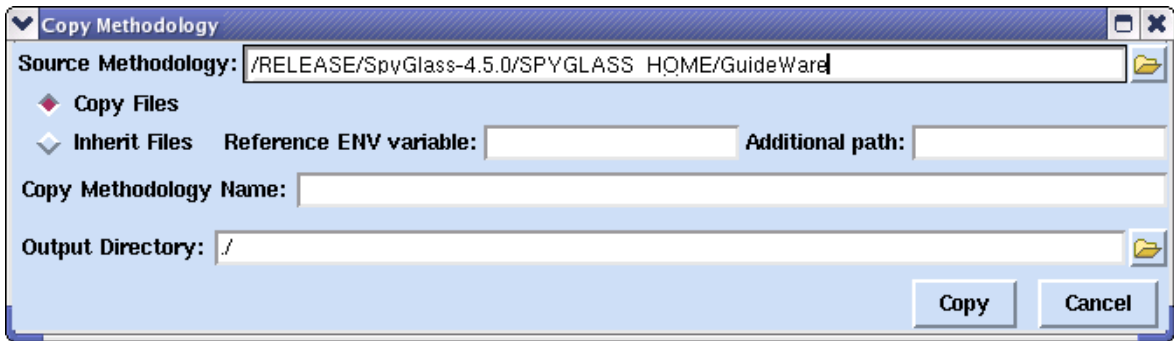



FIGURE 35. Copy Methodology

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.spg
```

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.

Atrenta Console 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:

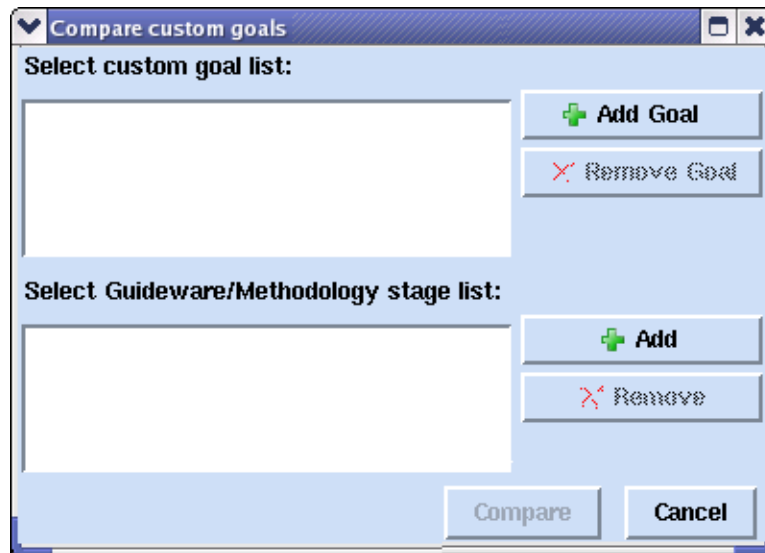


FIGURE 36. Compare Custom Goals

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:

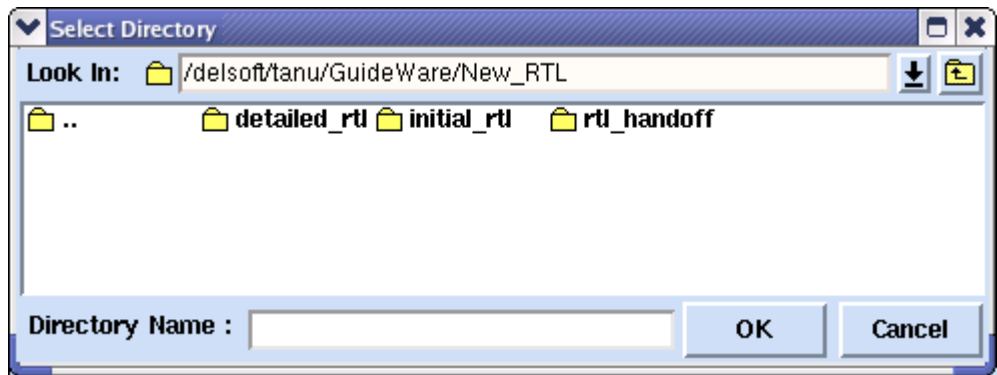


FIGURE 37. Select Directory

4. Click the *Compare* button.

The *Goal Comparison Summary* dialog appears, as shown in the following figure:

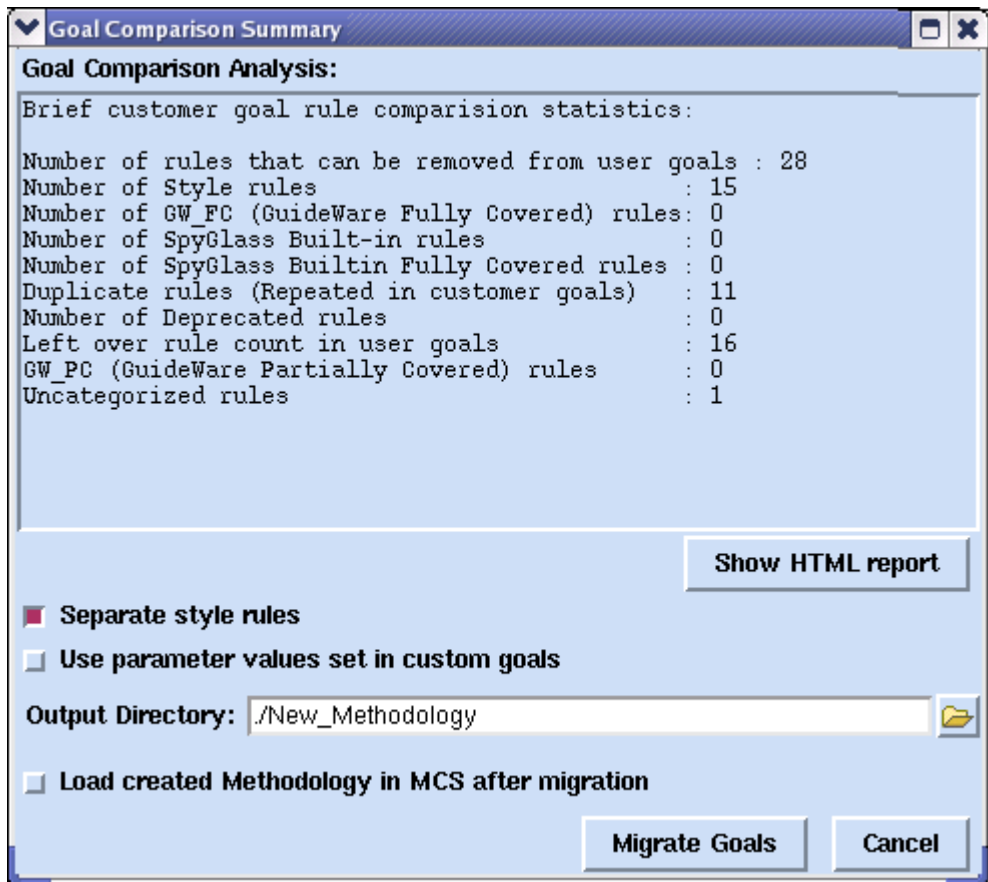


FIGURE 38. Goal Comparison Summary

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:

Migrating Custom Goals

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								
<input type="button" value="Show"/>	<input type="button" value="Hide"/>	<input type="button" value="Hide"/>	<input type="button" value="Hide"/>	<input type="button" value="Hide"/>	Total Rules	Total GW	Total GW Opt	Total Customer
Common	Common_GWOpt	GW_Only	GW_Opt_Only	Cust_Only				
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 expres				
GW_Only	rtl_handoff//audit/rtl_audit-vhdl.spg rtl_handoff//lint/synthesis-vhdl.spg	lint	AllocExpr	VHDL: Allocator expres				
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	AsqnOverflow-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 39. 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, Atrenta Console 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:

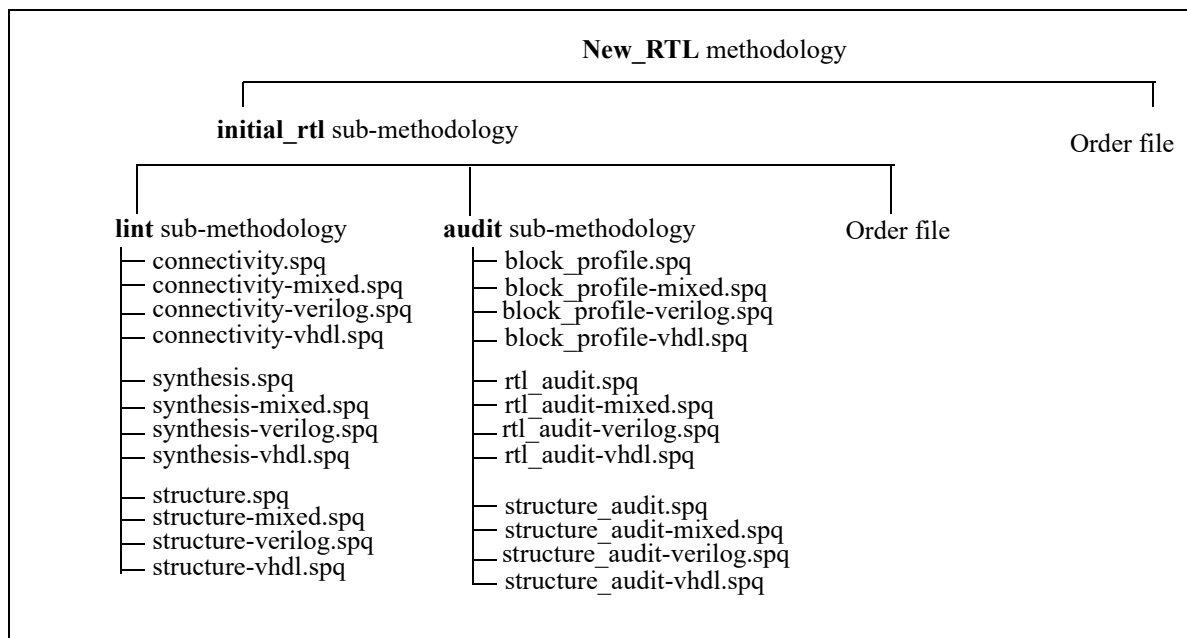


FIGURE 40. Sample Directory Structure

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*
  
```

Order File

```
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, Atrienta Console 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 Atrenta Console 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 Atrenta Console 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_rtl` goal in GuideWare 2.0 represents following four goals in GuideWare 1.0:

- connectivity
- simulation
- synthesis
- structure

Working with SpyGlass Design Constraints

Overview

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 Atrenta Console 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 `hdl1libdu` 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           // Contents of constraint_include.sgdc
include constraint_include.sgdc          current_design test
current_design and_new                   test_mode -name test.w1 -value 1
test_mode -name in1 -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, Atrenta Console 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"`, `"\myvhdsig#11\"`. You only need to escape the double quote character in an escaped name as in `"\myvlogsig\"23\""`, `"myvhdsig\"5\""`.*

NOTE: *You can also use Synopsys-style escaped names by specifying the following command in Atrenta Console 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.

Atrenta Console 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

Conditionally Specifying SGDC Constraints

```
# 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

```


Conditionally Specifying SGDC Constraints

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
```

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.*

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, sta_scg, strict_sta
Default value	sta_compliant
Example	

<i>Console/Tcl-based usage</i>	<code>set_parameter sdc_domain_mode strict</code>
<i>Usage in goal/source files</i>	<code>-sdc_domain_mode=strict</code>

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 `sta_scg` or `strict_sta`, 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	
<i>Console/Tcl-based usage</i>	<code>set_option sdc_generate_cfp yes</code>
<i>Usage in goal/source files</i>	<code>-sdc_generate_cfp=yes</code>

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 1. 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.

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.

 Converting SDC Attributes into SGDC Commands

Used by	sdcs2sgdc flow
Options	yes, no
Default value	no
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter enable_generated_clocks yes</code>
<i>Usage in goal/source files</i>	<code>-enable_generated_clocks =yes</code>

Used by	sdcs2sgdc flow
Options	yes, no
Default value	no
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter sdc_generated_clocks yes</code>
<i>Usage in goal/source files</i>	<code>-sdc_generated_clocks=yes</code>

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_uncertainty` 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 `set_clock_sense` SDC command is converted to the `clock_sense` SGDC command during the `sd2sgdc` 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 is 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 model U1
```

Following is the converted SGDC constraint:

```
set_lib_timing_mode -modes model -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, Atrienta Console 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, Atrenta Console 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, Atrenta Console 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, Atrenta Console 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>`.
- Atrenta Console 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:

Importing Block-Level SGDC Commands to Chip-Level

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.

To generate hierarchical SGDC file(s), perform the following steps:

1. Click the *Add Design Files* tab.
2. Specify the design files.
3. Specify the migration file containing commands for importing block-level SGDC file(s) to chip-level.

For details, see [Creating a Migration File](#).

4. Click the *Set Read Options* tab.
5. Set the value of the *Hierarchical SGDC Modes* option to *Generation Mode*.
6. Click the *Run Design Read* tab.
7. Click  **Generate Hierarchical SGDC**.

The above step generates SGDC file(s) containing the imported SGDC commands. For details, see [Generated Hierarchical SGDC File\(s\)](#).

Alternatively, you can specify the following command in the project file to generate hierarchical SGDC file(s):

```
set_option gen_hiersgdc yes
```

Generated Hierarchical SGDC File(s)

Atrenta Console 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

Atrenta Console 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: *Atrenta Console 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 hierarchical SGDC file(s), perform the following steps:

1. Click the *Set Read Options* tab.
2. Set the value of the *Hierarchical SGDC Modes* option to *Validation Mode*.
3. Click the *Run Design Read* tab.
4. Click  **Validate Hierarchical SGDC**.

After performing the above steps:

1. SpyGlass first checks for the migrated SGDC files after [Generating Hierarchical SGDC File](#) in the `gen_hiersgdc/spyglass_reports/imported_sgdc` directory.
2. SpyGlass then validates the migrated SGDC files with respect to the input block-level SGDC files.

To validate the generated SGDC file, you do not have to specify the migrated SGDC files generated in the [Generating Hierarchical SGDC File](#) step.

You can perform this task by specifying the following command in the project file:

```
set_option validate_hiersgdc yes
```

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 (`set_option gen_hiersgdc yes`), validation mode (`set_option validate_hiersgdc yes`), and SpyGlass CDC abstraction flow.*

Implementing Scoping in SGDC Commands

Atrenta Console 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, Atrenta Console 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, Atrenta Console 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. 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
```

Implementing Scoping in SGDC Commands

```

set_case_analysis -name top.bil.mi2.in -value 0
set_case_analysis -name top.bil.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.

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.mil.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.mil.in -value 1
current_design top
set_case_analysis -name top.bil.mil.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.mil.in -value 1
current_design top
set_case_analysis -name block::mil.in -value 1
set_case_analysis -name top.bil.mil.in -value 0
```

Here, all the block-level commands are ignored. For example, `mil` of `M` inside `block` is ignored.

Handling SystemVerilog Objects in SGDC

Atrenta Console 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:

Handling SystemVerilog Objects in SGDC

```

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;
```

Handling SystemVerilog Objects in SGDC

```

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_label>[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

Overview

Atrenta Console displays all violation messages of the currently loaded goal in the *Results* pane, as shown in the following figure:

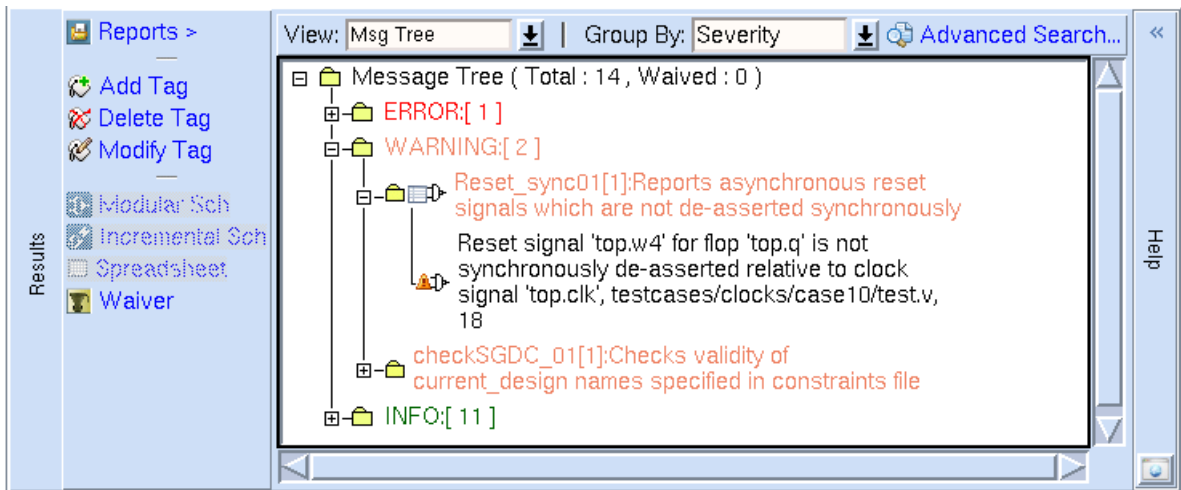


FIGURE 1. Violation Messages for Currently Loaded Goals

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 2. Run Goal

Based on the format in which you want to view messages, click the *View* drop-down list and select the required format, such as *Msg Tree*, *Msg Summary*, *Module Hierarchy*, and *Waiver Tree*. For details on these formats, refer to *Atrenta Console Reference Guide*.

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:

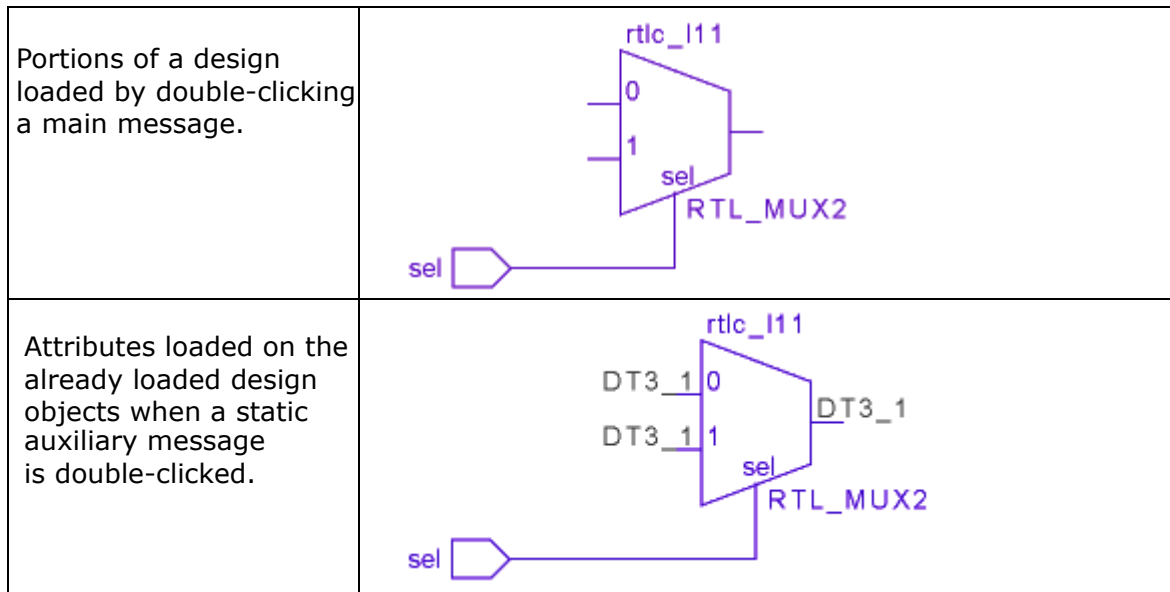


FIGURE 3. Static Auxiliary Messages

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:

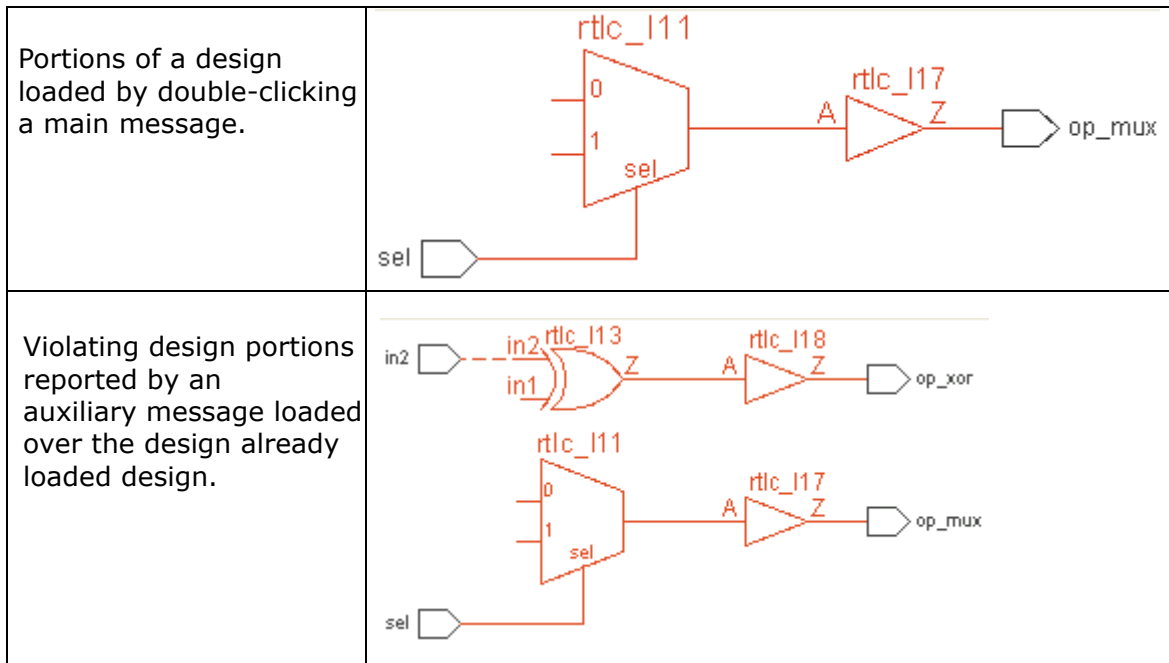


FIGURE 4. Non-Static Auxiliary Messages

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, Atrenta Console 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, Atrenta

Console 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

Atrenta Console 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* (.SWI 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 (.swl 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 (.swl file).
- By using the *Waiver Editor* window
Specify details in the appropriate fields in the *Waiver Editor* window. Based on the details specified, Atrenta Console generates corresponding `waive` constraints in a waiver file.
You can use this window to create or modify waiver files.
For details on this window, refer to the *Waiver Editor* topic of *Atrenta Console Reference Guide*.

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
```

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.swl
waive -rule r1

current_goal G1 -alltop

waive -rule r2
set_goal_option default_waiver_file goal.swl
waive -rule r3

save_project

-----
$ cat project.swl
waive -rule { {r1} }
waive -rule { {r2} }

$ cat goal.swl
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:

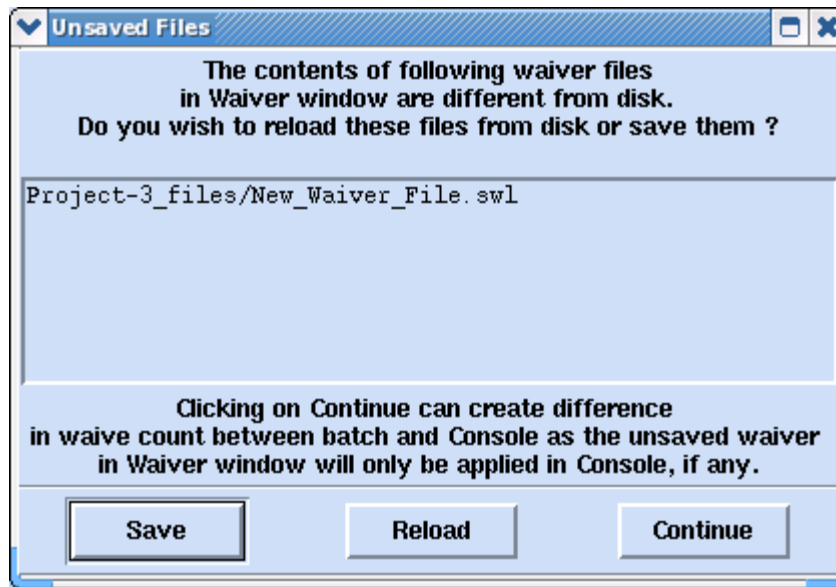


FIGURE 5. Unsaved Files

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                                     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

Atrenta Console 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
```

Atrenta Console 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 in either 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](#).

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](#) (.swl). These waiver files appear in the left-most section of the Waiver Editor window.

The following figure shows the *Waiver Editor* window:

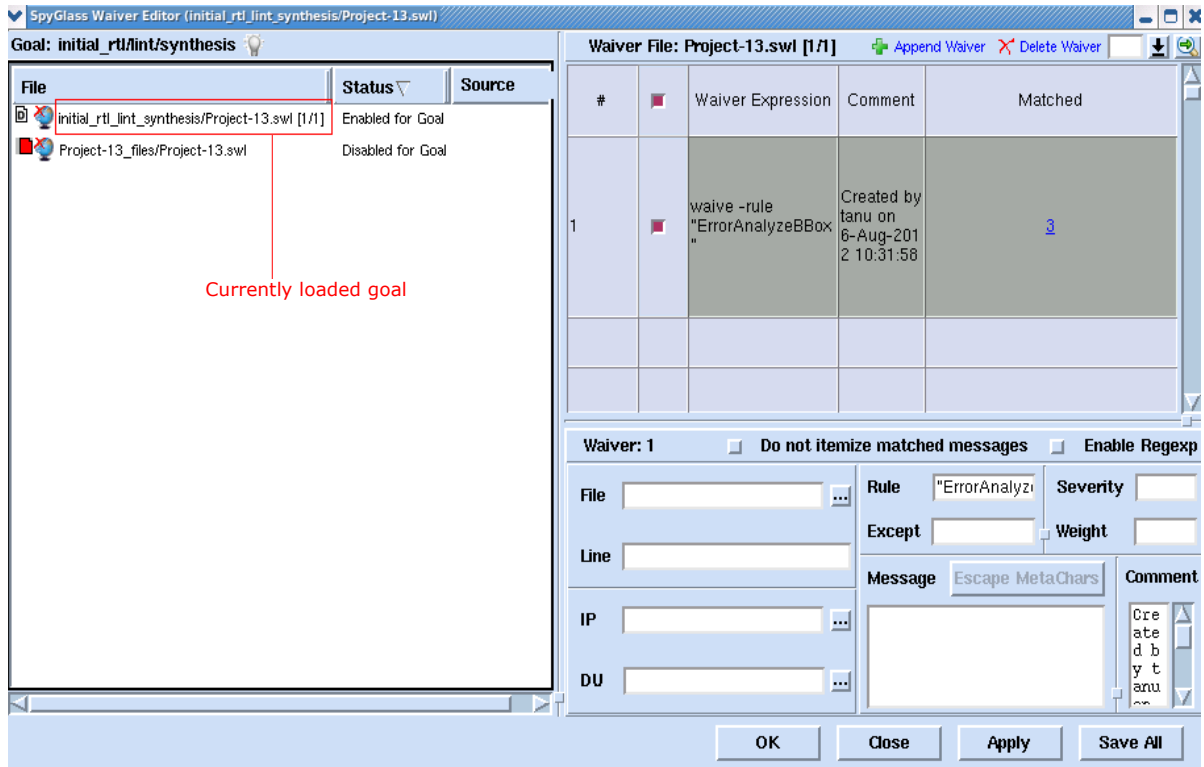



FIGURE 6. Waiver Editor Window

In the above window, you can add new or existing waiver files for the currently loaded goal. For details on adding new or existing waiver files, refer to the description of the Add File and New File options in the Right-Click Options of Tree-View Section topic of Atrenta Console Reference Guide.

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, refer to the *Waiver Editor Window* section of *Atrenta Console Reference Guide*.

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* page in the *Results* pane. For details on this page, refer to *Atrenta Console Reference Guide*.

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.

Option in the Sub-Menu	Description
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:

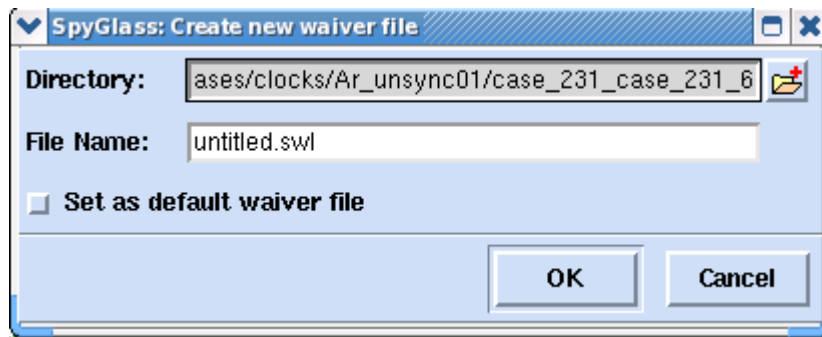


FIGURE 7. Create New Waiver File

In the above dialog, specify the name of the file and select the *Set as default waiver file* option.

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 ] [ -regex ] [ -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:

Waiving Messages

Argument	Description
-file_lineblock	<p>Use this argument to waive messages for a block of lines in a source file.</p> <p><file-sline-eline> is a space-separated tuple of source file name, start line number, and end line number in the following format:</p> <p><file-name> <line1> <line2></p> <p>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>.</p> <p>Note: Use multiple -file_line/-file_lineblock arguments, each with one argument.</p> <p>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.</p>
-file_line	<p>Use this argument to waive rule messages for a particular line of a source file.</p> <p><file-line> is a space-separated pair of source file name and line number in the following format:</p> <p><file-name> <line-num></p> <p>Note: Use multiple -file_line/-file_lineblock arguments, each with one argument.</p> <p>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.</p>
-file	<p>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.</p>

Argument	Description
-du and -ip	<p>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.</p> <p>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.</p> <p><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:</p> <ul style="list-style-type: none"> • 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) <p>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.</p> <p>Note the following points:</p> <ul style="list-style-type: none"> • 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 <p>By default, SpyGlass waives violations on only those module instances that are present in the IPs specified by the waive -ip specification.</p>
-rule/-rules	<p>Use these arguments to waive messages of the specified rules, rule groups, or products or by rule type keywords.</p> <p><rule-list> refers to a space-separated list of rule names, rule group names, or product mnemonics.</p> <p>This argument is case-sensitive.</p>

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.
-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>.
-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.

Argument	Description
-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.

Waiving Messages

- 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, Atrienta Console 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, refer to the *Wildcard Mode* topic of *Atrenta Console Reference Guide*.

Processing Regular Expressions

SpyGlass uses the `regcomp` and `regexexec` 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 `regexexec` 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 '.*'" -regex
```

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),
\[Hierarchy:srtop\]" -regex
```

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 1 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 1 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 [\$]

Waiving Messages

	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.	\ or []
.	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.	\. or [.]
\	To use a literal backslash symbol within a regular expression, the backslash symbol must be escaped with a backslash character or contained inside brackets.	\\ or [\\]
~`!@#%&_={};:'",<>	The remaining symbol characters are treated as literal characters and do not need to be escaped within regular expressions.	
(space)	In regular expressions, the space character is ignored except when it appears inside a character class or is preceded by a backslash.	[] or \
{}	To use a literal left or right curly bracket within a regular expression, the curly bracket symbol must be contained inside brackets.	[{] [}]
()	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.	\(and \) or [(] and [)]

Selective Use of Regular Expressions

When you specify the `-regex` 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 `-regex` argument.

Conversely, you may want to process the values of some arguments as normal values even when the `-regex` 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 `-regex` 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 `-regex` 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 -regex -file ".*" ...
```

or

```
waive -regex -du ".*" ...
```

- Specify the following command to waive messages in all files whose names start with `lib`:

```
waive -regex -file "^lib.*" ...
```

- Specify the following command to waive messages in all design units whose names end with `vwe`:

```
waive -regex -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.

Atrenta Console 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, Atrenta Console 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 swl 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 `wave -import` constraint, as shown below:

```
wave -import b1 b1.swl -disable
```

The above specification will disable the `wave -import` command.

NOTE: *Only the `-disable` and `-comment` arguments are supported with `wave -import` constraint. No other argument is supported with `wave -import` constraint.*

- File names in the imported `wave -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 `wave` 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 `wave` 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

Waiving Messages

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
```

Atrenta Console considers the above specification as two different pragmas. However, if the starting directive is a comment or a non-waiver pragma, Atrenta Console 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 ←———— rule1 active
...
//spyglass disable_block rule1 ←———— rule1 waived
...
//spyglass enable_block rule1 ←———— rule1 still waived
...
//spyglass enable_block rule1 ←———— rule1 still waived
...
//spyglass enable_block rule1 ←———— rule1 active
...

```

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 ←———— rule1 active
...
//spyglass enable_block rule1 ←———— rule1 waived
...
//spyglass enable_block rule1 ←———— rule1 active
...
//spyglass enable_block rule1 ←———— rule1 still active
...

```

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

Waiving Messages

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 ← rule1 active
...
//spyglass disable_block rule1 ← rule1 waived
...
//spyglass enable_block rule1 ← rule1 waived
...
//spyglass enable_block rule1 ← rule1 active
...

```

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 ← rule1, rule2, rule3 active
...
//spyglass enable_block rule1 ← rule1, rule2, rule3 waived
...
//spyglass enable_block rule2 rule3 ← rule1 active;
...
//spyglass enable_block rule2 rule3 ← rule2, rule3 waived
...
//spyglass enable_block rule2 rule3 ← 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 Atrenta Console 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 ← rule1 active
...
//spyglass disable_block rule1 ← rule1 waived
...
//spyglass enable_block rule1 ← rule1 still waived
...
//spyglass enable_block rule1 ← rule1 still waived
...
//spyglass enable_block rule1 ← rule1 active
...

```

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:

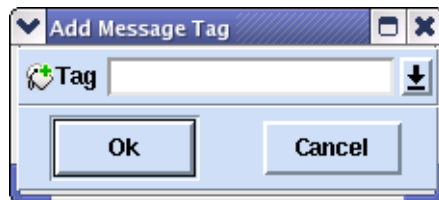


FIGURE 8. Add Message Tag

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.

Modifying a Tag

To modify the tag for a rule message, perform the following steps:

1. Select the message appearing in the *Results* pane.
2. Select the *Modify Tag* option in the right-most bar in the *Results* pane.

Alternatively, right-click on the message and select the *Tag -> Modify* option from the shortcut menu.

The *Modify Message Tag* dialog appears with the selected message and its tag, as shown below:

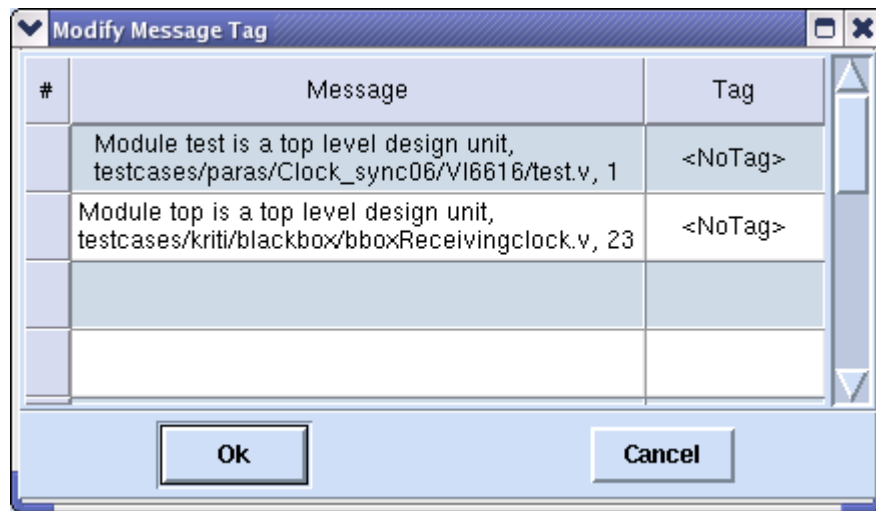


FIGURE 9. Modify Message Tag

3. In the above dialog, select the tag name in the *Tag* field. A drop-down list appears containing all the available tags.
4. Select the required tag from the drop-down list or specify your own tag in the *Tag* textbox.
5. Click the *OK* button to apply the modified tag to the selected message.

NOTE: *The Tag -> Modify shortcut menu option is enabled only if you have added a tag to a rule message.*

NOTE: *You can also add/delete/modify tags for more than one message at a time by selecting multiple messages (either by dragging the mouse pointer across the messages or by individually clicking the messages while holding down the <Ctrl> key on the keyboard) and applying the tag settings as described above.*

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.

Working with Aggregated Reports

Overview

This chapter describes the following reports generated in Atrenta Console:

- *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, Atrenta Console 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, Atrenta Console 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, Atrenta Console 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, Atrenta Console 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 Atrenta Console 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:

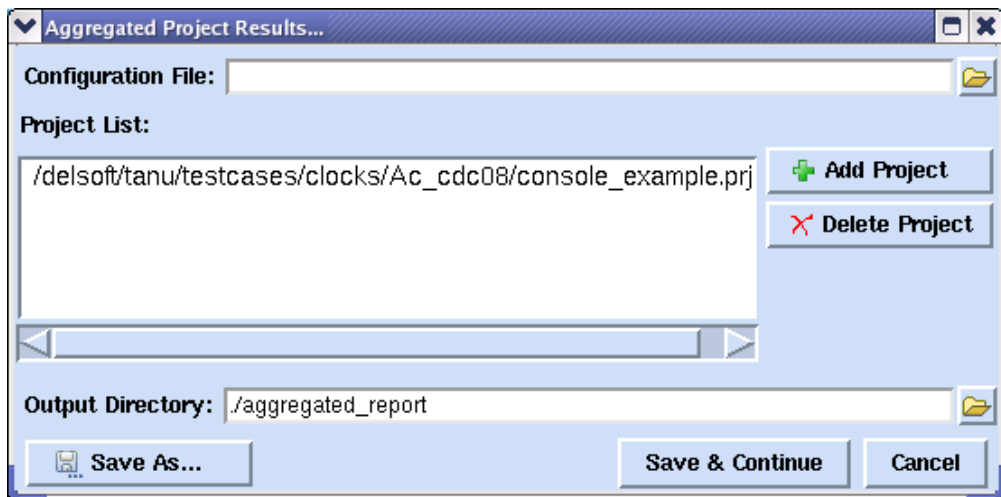



FIGURE 1. Aggregated Project Results


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, Atrenta Console creates

Generating Aggregated Project Results

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 () 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:

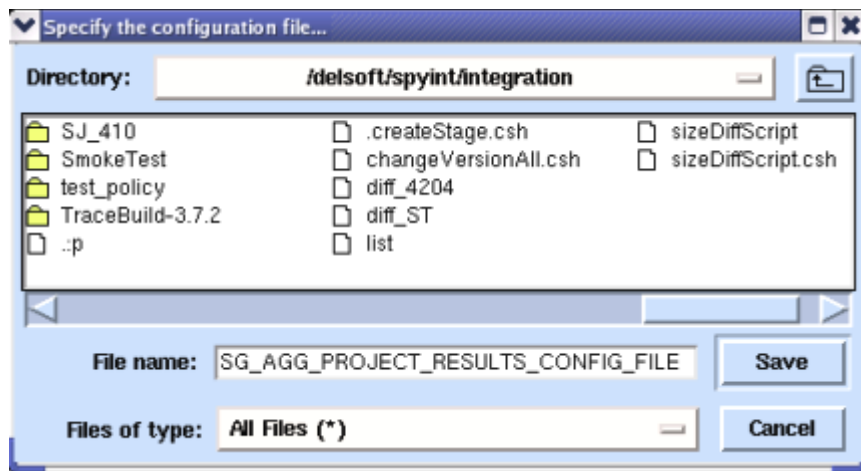


FIGURE 2. Specify the Configuration File

By default, Atrenta Console 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 Results* dialog appears displaying the location where the results are saved, as shown in the following figure:

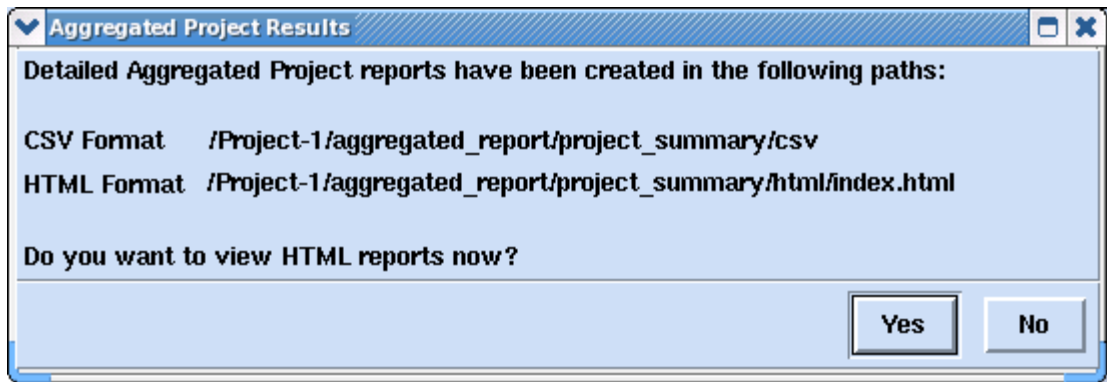


FIGURE 3. Aggregated Project Results Dialog Box

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).

NOTE: Refer to the *Atrenta Console Reference Guide* for details about the menu options.

- 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:

Generating Aggregated Project Results

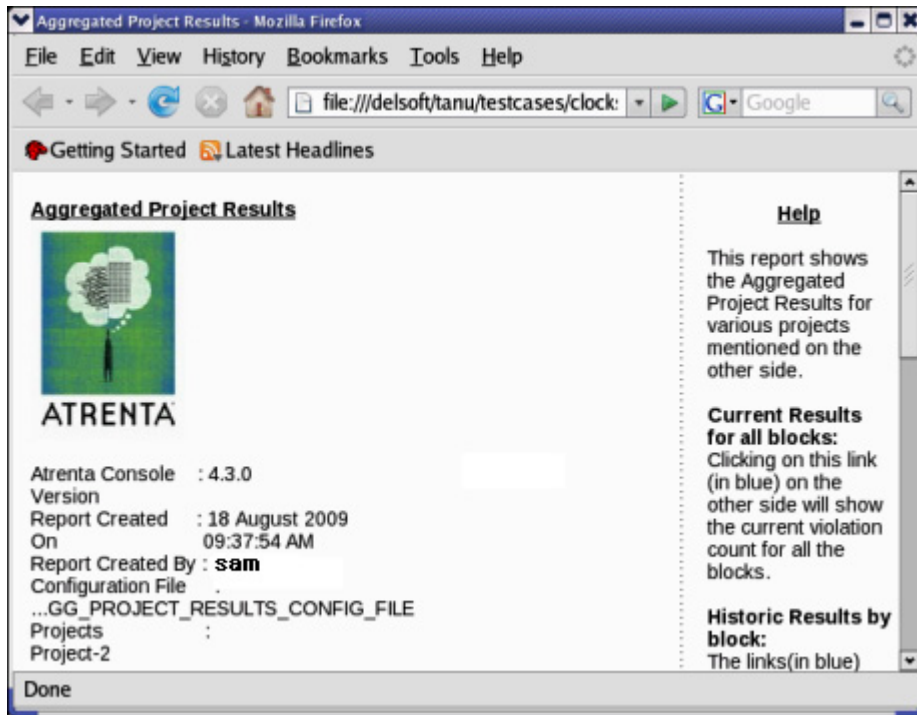


FIGURE 4. Aggregated Project Results Report

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:

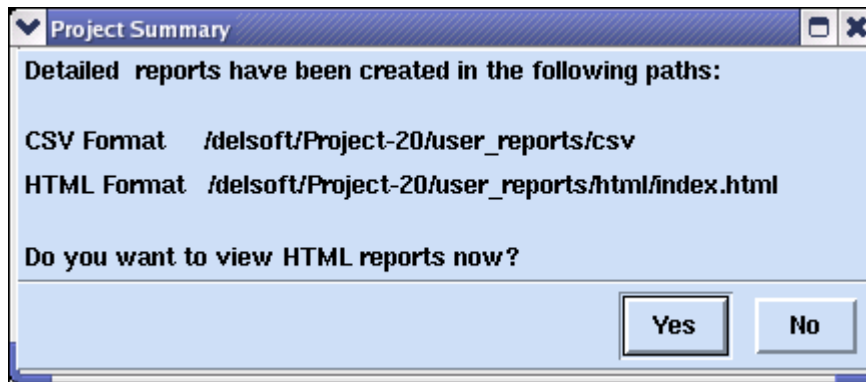


FIGURE 5. Project Summary Dialog Box

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, Atrenta Console 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

Atrenta Console 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).

NOTE: Refer to the *Atrenta Console Reference Guide* for details about the menu options.

- 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:

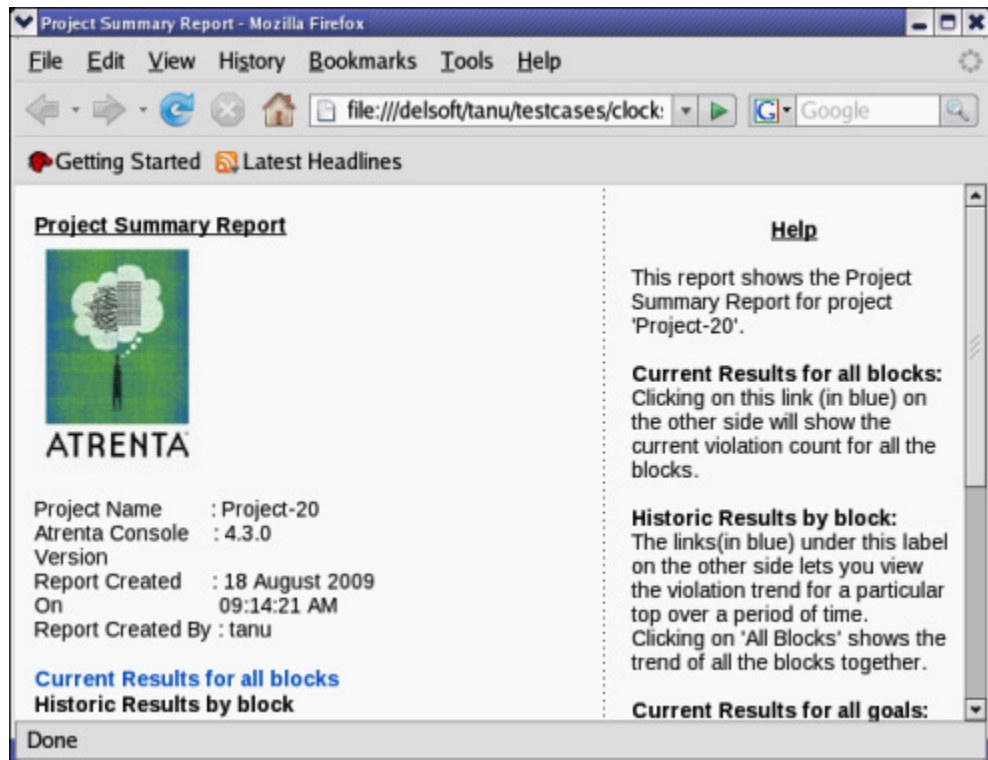


FIGURE 6. Project Summary Report

The HTML browser is divided into two sections. The left section of the browser displays information, such as the project name, the Atrenta Console 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.

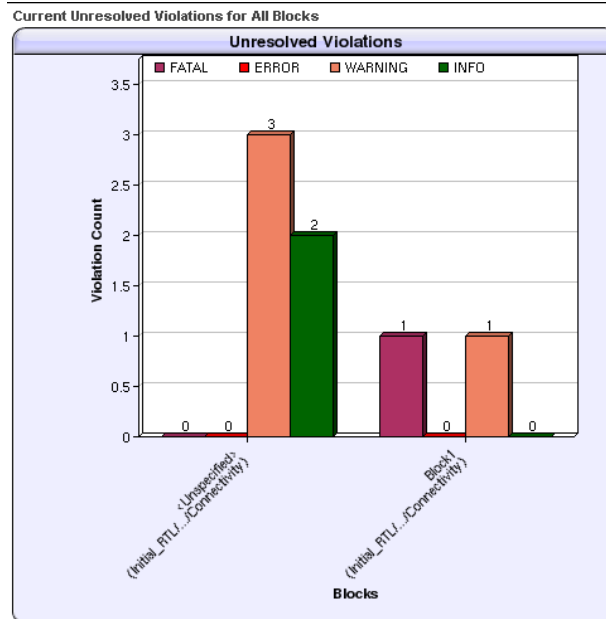


FIGURE 7. Unresolved Violations Chart

- 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.

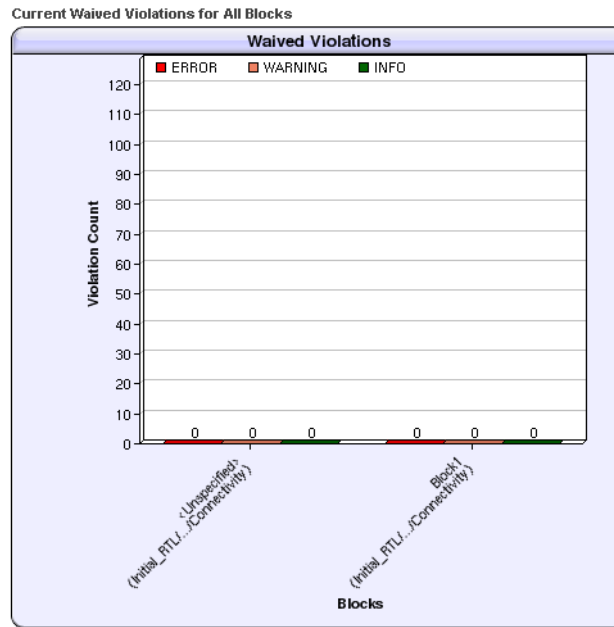


FIGURE 8. Current Waived Violations for All Blocks Chart

- 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 9. 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:

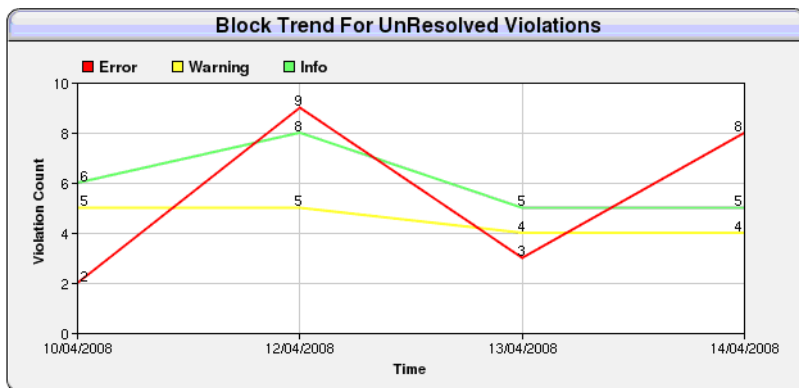


FIGURE 10. Block Trend for Unresolved Violations

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).

NOTE: Refer to the *Atrenta Console Reference Guide* for details about the menu options.

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 11. 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:

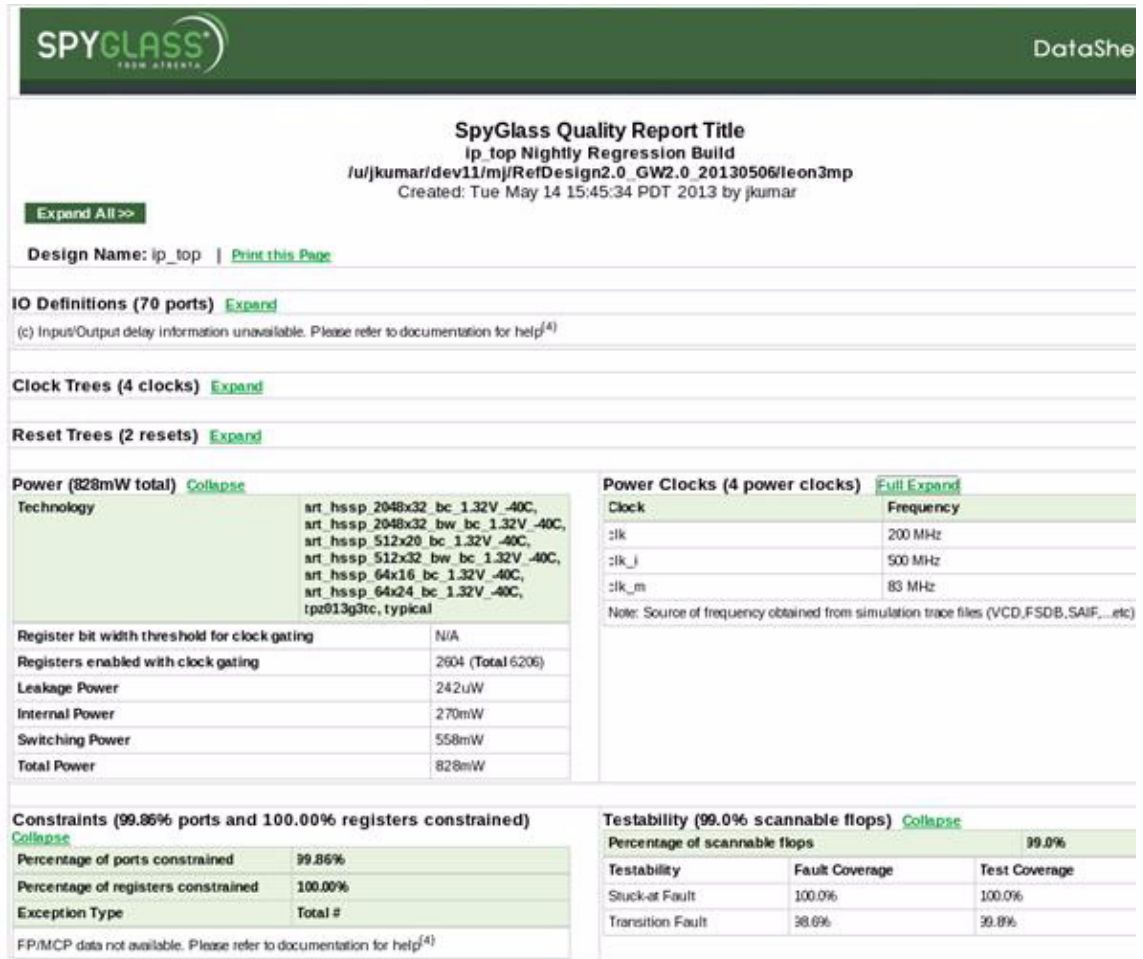


FIGURE 12. The DataSheet Report

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 Atrenta 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, Atrenta Console displays the *Datasheet Report* dialog, as shown in the following figure:

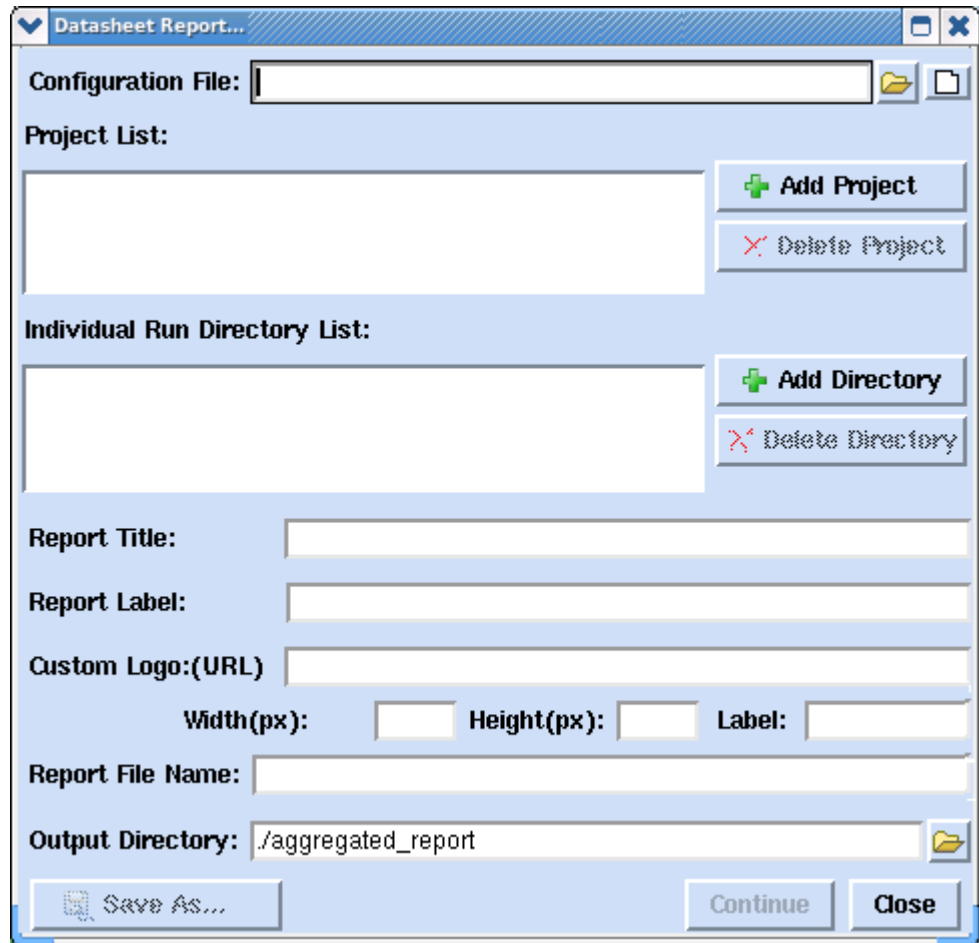



FIGURE 13. Configuring DataSheet Report

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, Atrenta Console 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:



FIGURE 14. 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 Atrenta Console 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. Atrenta Console considers this option only if you have NOT specified the -config_file option. If you specify both the -config_file and -project options, Atrenta Console 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
```

The DataSheet Report

```

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.

For timing, congestion, and more accurate design statistics, run the SpyGlass Physical methodology. For details, refer to the SpyGlass Physical Methodology and its rule documentation.

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: <ul style="list-style-type: none"> ● 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: <ul style="list-style-type: none"> ● Ac_sync01 ● Ac_sync02 ● Ac_unsync01 ● Ac_unsync02 <p>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.</p>	SpyGlass CDC Solution
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

The DataSheet Report

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	PHY_GateCount	SpyGlass Physical Solution
	PHY_GateArea	SpyGlass Physical Solution
	PHY_PhysicalSummary (or) Audit4Dump	SpyGlass Physical Solution (or) SpyGlass audits Solution
Timing	PHY_ClockDetail	SpyGlass Physical Solution
Congestion	PHY_CongModules	SpyGlass Physical 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 *DataSet* report:

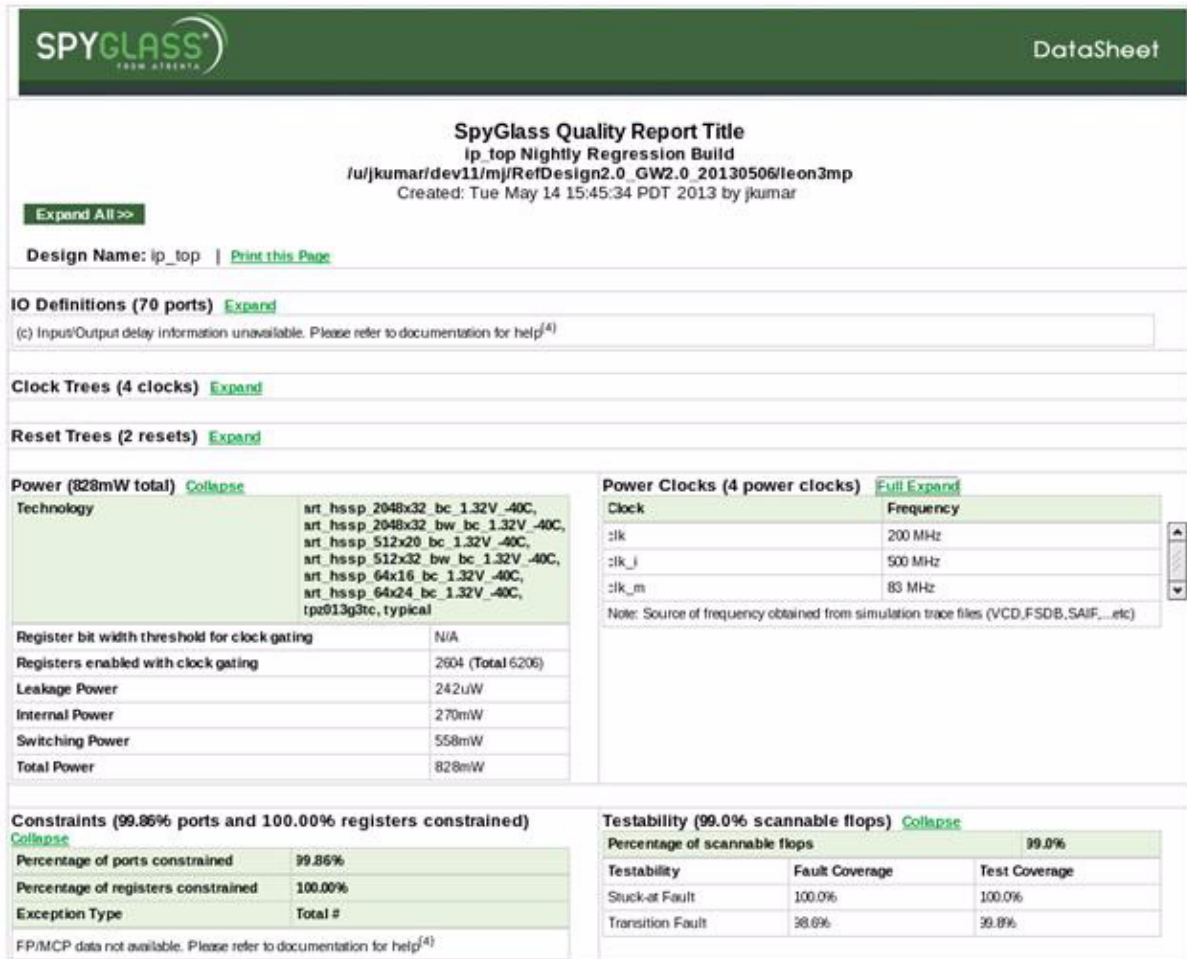


FIGURE 15. Sample DataSheet Report

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.

The DataSheet Report

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	Timing	Congestion	

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 16. 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, <code>datasheet_io_audit</code>
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	55.8235 MHz	AHB_WISHBONE	30	1169	0	0	0	1169	0	0	0	0	0	SDC/SGDC

FIGURE 17. DataSheet Report - Clock Trees

Atrenta Console 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 18. DataSheet Report - Reset Trees

Atrenta Console 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 <code>preset</code> or <code>clear</code>
Synchronous	<p>Active High: Specifies the number of flip-flops that use it as an active high synchronous reset</p> <p>Active Low: Specifies the number of flip-flops that use it as an active low synchronous reset</p>
Asynchronous	<p>Active High: Specifies the number of flip-flops that use it as an active high asynchronous reset</p> <p>Active Low: Specifies the number of flip-flops that use it as an active low asynchronous reset</p>

Power

A sample of the *Power* section is shown in the following figure:

The DataSheet Report

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	1.04mW
Switching Power	1.21uW
Total Power	1.56mW

FIGURE 19. DataSheet Report - Power

Atrenta Console 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 20. DataSheet Report - Power Clocks

Atrenta Console 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 #
FPMCP data not available. Please refer to documentation for help (4)	

FIGURE 21. DataSheet Report - Constraints

Atrenta Console 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	85.8%	91.2%

FIGURE 22. DataSheet Report - Testability

Atrenta Console 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 Collapse	
Statistic	Count
Synthesizable gates (NAND2 equivalent)	70752
Total area	0.1585 mm ²
Registers	4018
Latches	318
Tristates	0

FIGURE 23. DataSheet Report - Design Statistics

The above section displays a table showing different types of statistic data and their counts.

In this table, the first two statistics data are populated from the SpyGlass Physical flow, if it exists. Otherwise, the corresponding cells appear blank.

The rest of the three statistics are populated from the SpyGlass Physical flow, if it exists. Otherwise, as a second priority, the Audit flow data is populated and displayed, if it exists.

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. For RTL logic, standard cell utilization specified in SpyGlass Physical goal is used (default is 60%).
Registers	Specifies the total number of flip-flop instances in the design.

The DataSheet Report

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

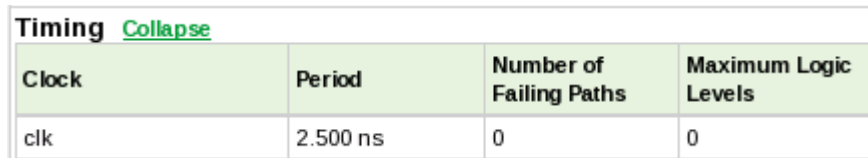
This section displays the number of ports for each black box. The details of each field of this table are described below:

Field Name	Description
Black Box Name	Specifies the name of the black box
Number of Ports	Specifies the number of ports

Timing

This section displays the timing data populated from the SpyGlass Physical flow.

A sample *Timing* section is shown in the following figure:



Timing Collapse			
Clock	Period	Number of Failing Paths	Maximum Logic Levels
clk	2.500 ns	0	0

FIGURE 24. DataSheet Report- Timing

The details of each field of this table are described below:

Field	Description
Clock	Specifies all primary and derived clocks defined in an SDC file.
Period	Specifies a clock period.

Number of Failing Paths	Specifies the total number of gross failing path groups, when grouped by a common source bus and a common destination bus for the specified clock. It means multiple paths from different bits in a source bus to different bits in a destination bus are counted as 1. The threshold for gross failing paths is based on the <i>PHY_ClockDetail</i> parameter. When no valid timing paths exist for a given clock, "-" appears in the corresponding cell.
Maximum Logic Levels	Specifies the maximum number of standard cell instances as analyzed across all timing paths of the specified clock. When no valid timing paths exist for a given clock, "-" appears in the corresponding cell.

Congestion

This section displays the congestion data populated from the SpyGlass Physical flow.

A sample *Congestion* section is shown in the following figure:

Congestion Collapse		Standard Cell Count	Internal Congestion Score	Peripheral Congestion
Module Name	Hierarchical Instance Name			
tpcc_piu_GlueLogic	l_tpcc_piu	10112	10.00	High
tpcc_piu	l_tpcc_piu	13403	7.10	High
tpcc_core	l_tpcc_core	26939	4.97	Low

FIGURE 25. DataSheet Report - Congestion

The details of each field of this table are described below:

Field	Description
Module Name	Specifies the RTL module name of a grossly congested module instance.

The DataSheet Report

Hierarchical Instance Name	Specifies the full hierarchical instance name of a grossly congested RTL module instance. In the table, only leaf-level name appears. Place the cursor over this name to see the complete path in a tool-tip.
Standard Cell Count	Specifies the number of standard cell instances in a congested RTL module instance.
Internal Congestion Score	Specifies the SpyGlass Physical internal congestion score of the RTL module on a scale of 0-10. The following are some scores and their meanings: <ul style="list-style-type: none">● 8-10: Specifies a grossly congested module that requires very low utilization during place and route.● 7-8: Specifies a congested module that requires low utilization during place and route.● 0-7: Specifies a non-congested module.
Peripheral Congestion	Specifies a SpyGlass Physical grade for external connectivity of the specified module relative to its size. A module with a high or a very high grade may contribute to congestion in the parent module.

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:

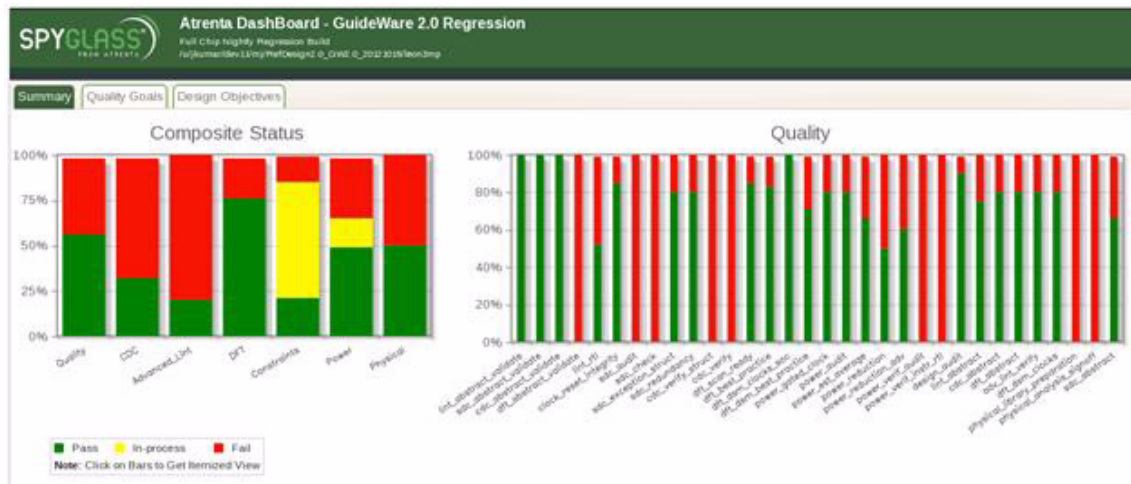


FIGURE 26. The Dashboard Report

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](#).

The Dashboard Report

A sample individual module *Dashboard* report is shown in the following figure:



FIGURE 27. 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 Atrenta 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. Atrenta Console considers this option only if you have NOT specified the -config_file option. If you specify both the -config_file and -project options, Atrenta Console 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:

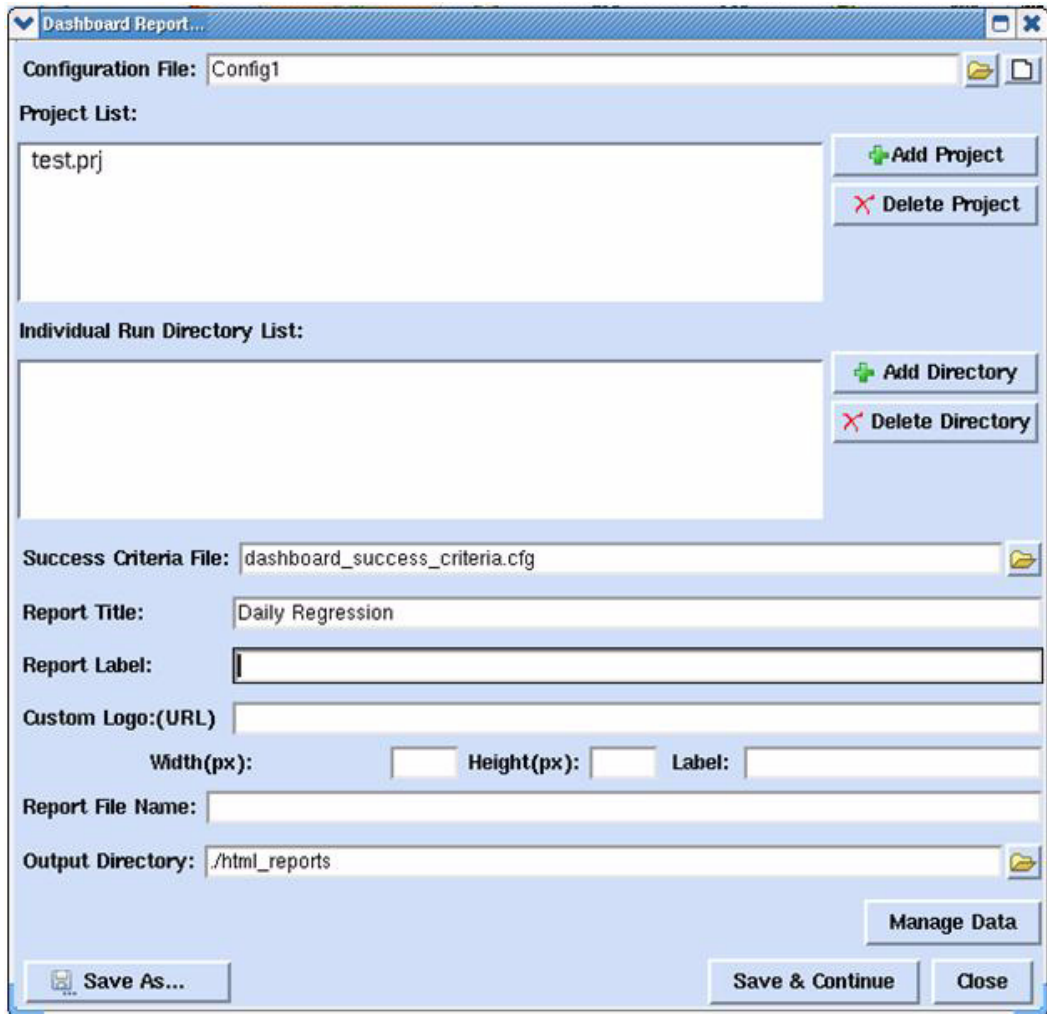


FIGURE 28. Configuring Dashboard Report

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, Atrenta Console 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:



FIGURE 29. 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_DASHBOARD <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/auxiliary/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*.

The DashBoard Report

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	<p>Specifies the success criteria. For example, you can specify the following criteria for the Power design objective:</p> <pre>-criteria {switching_power<50uW, total_power<80uW,leakage_powe<20nW}</pre> <p>For information on other supported items, see Variables Used in the Success Criteria File.</p> <p>Following operators are accepted in the mathematical expressions: <=, >= , >, <, !=, =</p> <p>The criteria that are mentioned as coverage/Percentage are percentage (%) numbers and the rest are product identified numeric values.</p> <p>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.</p> <p>Following units are supported in unit conversion:</p> <ul style="list-style-type: none"> ● 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) <p>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 <code>display_only</code>, as shown in the following example:</p> <pre>set_design_objective Power -criteria {switching_power=display_only,total_power=display_only ,leakage_power=display_only}</pre>

Argument	Description
-block	Specifies a comma-separated list of blocks on which the specified criteria is applicable. If you do not specify this argument, Atrenta Console 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, Atrenta Console 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	<p>Specifies the criteria to qualify for the Pass status, as shown in the following example:</p> <pre>-severity {Fatal=0,Error=0,Warning<1000,Waived-Error=0}</pre> <p>Valid severities accepted in this category are based on SpyGlass reported severity class messages that can be waived. SpyGlass accepts following case-insensitive labels:</p> <ul style="list-style-type: none"> • Fatal, Error, Warning, Info (SpyGlass reported severity classes) • Waived-Error, Waived-Warning, Waived-Info (severity class messages that can be waived) <p>In the mathematical expressions, following operators are accepted:</p> <pre><=, >= , > , < , !=, =</pre> <p>The values set in these expressions are number of SpyGlass reported (or) waived severity message counts.</p>
-goal	<p>(Optional) Specifies a comma-separated list of goals on which the specified criteria should be applied. If you do not specify this argument, Atrenta Console applies the specified criteria on all goals. The following example shows the usage of this argument:</p> <pre>set_quality_criteria -severity {Error<5,Warning<1000,Waived-Error=0} -goal {goal1,goal2}</pre>
-top	<p>(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, Atrenta Console applies the specified criteria on all blocks. The following example shows the usage of this argument:</p> <pre>set_quality_criteria -severity {Error<5,Warning<1000,Waived-Error=0} -goal {goal1,goal2} -top {block1,block2}</pre>

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
```

The DashBoard Report

```
[ -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

The DashBoard Report

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.

The DashBoard Report

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}
```

The DashBoard Report

```

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

Atrenta Console 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:



FIGURE 30. 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%

The DashBoard Report

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:

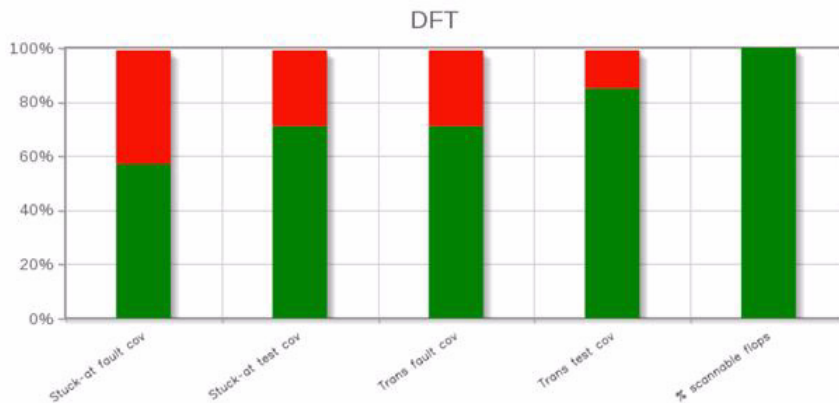


FIGURE 31. Product Specific Chart-Summary Tab

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:

The screenshot shows the 'Design Objectives' tab for 'Stuck-at fault coverage'. The table lists modules and their corresponding success criteria and status.

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 32. 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:

The screenshot shows the 'Quality Goals' tab. It displays a grid where rows represent modules and columns represent various quality goals. The status for each goal is indicated by a color: green for 'Pass' and red for 'Fail'.

Module	int	sdc	cdc	dt	int_rl	clock	sdc	sdc	sdc	sdc	sdc	cdc	dt	dt	dt	dt	power	power	power	power	power	power	power	power	desiga	int	cdc	dt	adv	dt
	absly	strata	abstrn	abstrn	rese	audc	check	except	redundancy	verify	verify	scan	ready	p	cl	be	gate	audc	est	redu	redu	veri	veri	audc	abstract	abstract	abstract	v	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	Pass	Pass	Pass	Pass	Pass	Pass	
ac97_top					Pass	Pass					Fail	Fail	Pass	Pass		Pass								Pass	Pass	Pass	Pass	Pass	Pass	
greth					Fail	Pass					Fail	Fail	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	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						
ahbiflag					Pass																			Pass						

FIGURE 33. 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.

The Dashboard Report

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:

The screenshot shows the SpyGlass dashboard for 'Atrenta DashBoard - GuideWare 2.0 Regression'. The 'Quality Goals' tab is active, and the goal 'lint/clock_reset_integrity' is selected. The table below displays the data for this goal across various modules.

Module Name	Quality Goals	Run Status	Unresolved			Waived		Success Criteria	Status
			fatal	error	warning	error	warning		
chiplop	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
groth	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
ion3cg	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 34. 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:



FIGURE 35. 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:

Module	CDC			Advanced_Lint			DFT				Constraints				Power				Physical			
	Umsync cov	Sync cov	Failures	Max complexity	Failures	Stuck at L	Stuck at L	Trans fail	Trans test	% scanabl	% ports constr	% log constr	# unvertf	# unvertf	Total power	Leakage power	Internal power	Switching (core)	Timing (pe)	Flag	Area	Cache
chip_top	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	Pass	No Data	No Data			Pass	Fail	Fail	Pass	Fail	Pass	Pass	Fail
lp_top	Fail	Fail	Pass	Fail	Fail	Pass	Pass	Pass	Pass	Pass	Fail	Pass			Pass	Pass	Pass	Pass				
leon3cg	Pass	Pass	Pass	Fail	Fail	Fail	Pass	Pass	Pass	Pass	Fail	Fail	No Data	No Data	Pass	Pass	Pass	Pass				
mc_top															Pass	Pass	Pass	Pass				
pcidma	Fail	Fail	Pass	Fail	Pass	Pass	Pass	Fail	Pass	Pass	Fail	Pass			Fail	Pass	Fail	Fail				
wb_subsystem	Fail	Fail	Pass	Fail	Fail	Pass	Pass	Fail	Fail	Pass	Pass	Pass			Pass	Pass	Fail	Pass				
shbctrl																						
shbtag																						

FIGURE 36. 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 32](#) 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:

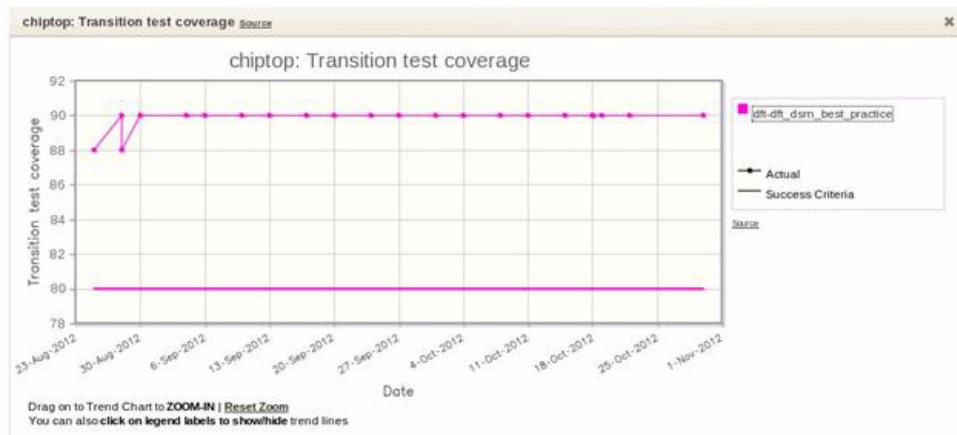


FIGURE 37. Trend Chart for Module/Design Objective

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:

The Dashboard Report

SPYGLASS Atrenta DashBoard - GuideWare 2.0 Regression
Full Chip Nightly Regression Build
/u/jkumar/Dev13/reg/HelDesign2_0_GW2_0_20121019/loc3reg

Summary | Quality Goals | Design Objectives | **chiptop**

Module: chiptop [Show All](#) | ■ Pass | ■ Fail

Quality Goals	Run Status	Unresolved		Waived		Success Criteria	Status
		total	error	warning	error		
Summary	Not completed	0	1059	2527	252	23	Failed goals = 11

[Show All](#) | ■ Pass | ■ Fail

Category	Design Objective	Success Criteria	Status

FIGURE 38. 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:

Summary | Quality Goals | Design Objectives | **chiptop**

Module: **chiptop** Show All | Pass | Fail

Quality Goals	Run Status	Unresolved			Waived		Success Criteria	Status
		fatal	error	warning	error	warning		
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 39. 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 37](#) for details of this trend graph.

The DashBoard Report

Module Dashboard-Design Objectives

Following figure displays the expanded view of the Design Objectives table in the Module Dashboard page:

SPYGLASS Atrenta DashBoard - GuideWare 2.0 Regression
Full Chip nightly Regression build
fujikumar/dev11/ny/RefDesign2_0_GW_0_20121015/leon3mp

Summary | Quality Goals | Design Objectives | **chiptop** | Show All | Pass | Fail

Module: chiptop

Quality Goals	Unresolved		Waived		Success Criteria	Status		
	Run Status	total	error	warning				
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_drm_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	
Power	Number of unverified MCP = NA	(Display only)	No Data	
	Total power = NA	Total power < 200mW	No Data	
	Leakage power = NA	Leakage power < 20mW	No Data	

FIGURE 40. 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:

<i>Category</i>	<i>Design Objectives</i>	<i>Success Criteria</i>
<i>Status</i>		

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 37* 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 Atrenta Console 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

Atrenta Console 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_integrity/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 41. 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:

The Dashboard Report

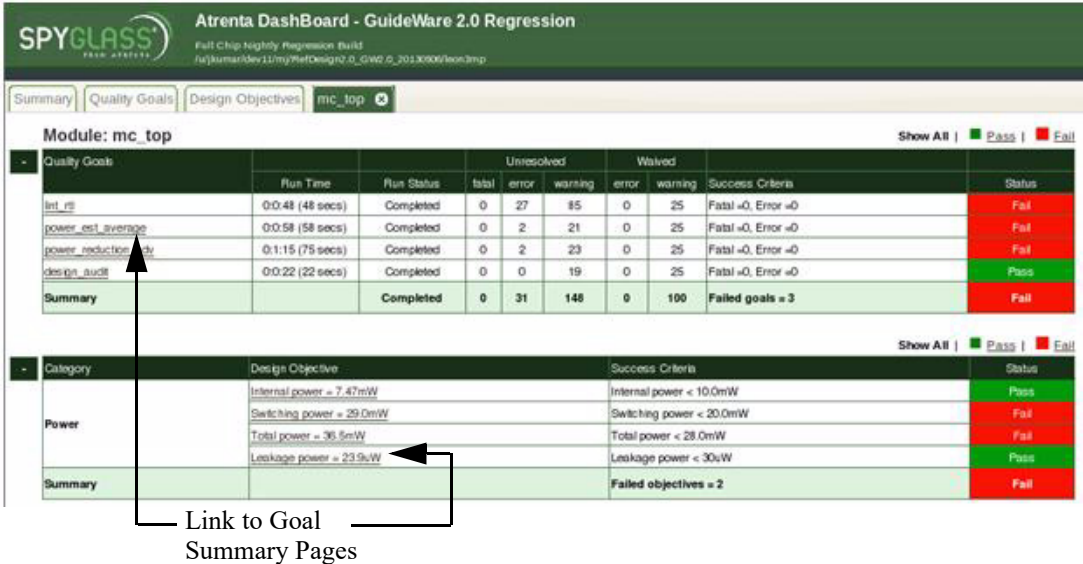
1. Select  in the row that you want to delete.
You can also choose all rows displayed in the above dialog by selecting  button in the column header.
2. Click the *Remove* button.

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:



The screenshot shows the SpyGlass dashboard for 'Atrenta DashBoard - GuideWare 2.0 Regression'. It displays two tables. The first table, 'Quality Goals', lists goals like 'int_rst', 'power_est_average', 'power_reduction_pct', and 'design_audit'. The 'power_est_average' goal is highlighted with a red status. The second table, 'Design Objectives', lists objectives like 'Internal power = 7.47mW', 'Switching power = 29.0mW', 'Total power = 36.5mW', and 'Leakage power = 23.9mW'. A red status is shown for 'Total power = 36.5mW'. A text box 'Link to Goal Summary Pages' has arrows pointing to the 'power_est_average' goal in the first table and the 'Total power = 36.5mW' objective in the second table.

Module: mc_top									
Quality Goals	Run Time	Run Status	Unresolved			Waived		Success Criteria	Status
			total	error	warning	error	warning		
int_rst	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_pct	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

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 = 23.9mW	Leakage power < 30uW	Pass
Summary		Failed objectives = 2	Fail

FIGURE 42. Creating Links to Reports

When you click the `power_est_average` link in the above report, the following goal summary page is displayed:

SPYGLASS 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 Fatais	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 43. 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/leon3xp/results/spyglass/mc_top
#   Report Report Location : : /u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20130506/leon3xp/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 noresimple report for any error        #
# reported.                                                                     #
#####
-----
Generated by rule      : PEPWR02
VCD file              : /u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20130506/leon3xp/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
                     : is_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 44. 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:

The screenshot shows the SpyGlass web interface for a goal summary. The header includes the SpyGlass logo, 'Atrenta Reports', 'Design Top: ip_top', and a 'Change Goal' dropdown menu set to 'cdc/cdc_verify'. The main content area is titled 'cdc/cdc_verify' and contains the following sections:

- 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
 - more/simple
 - more/simple_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 Fatafs	0 Errors	25 Warnings	0 Info
Reported Messages:		0 Fatafs	48 Errors	31 Warnings	26 Info
- Technology Summary: CDC(Advance CDC)**

Unynchronized crossings	=	33
Convergences	=	0

FIGURE 45. Sample Goal Summary

You can choose 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

Goal Summary

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 Atrenta Console as shown in the following figure:

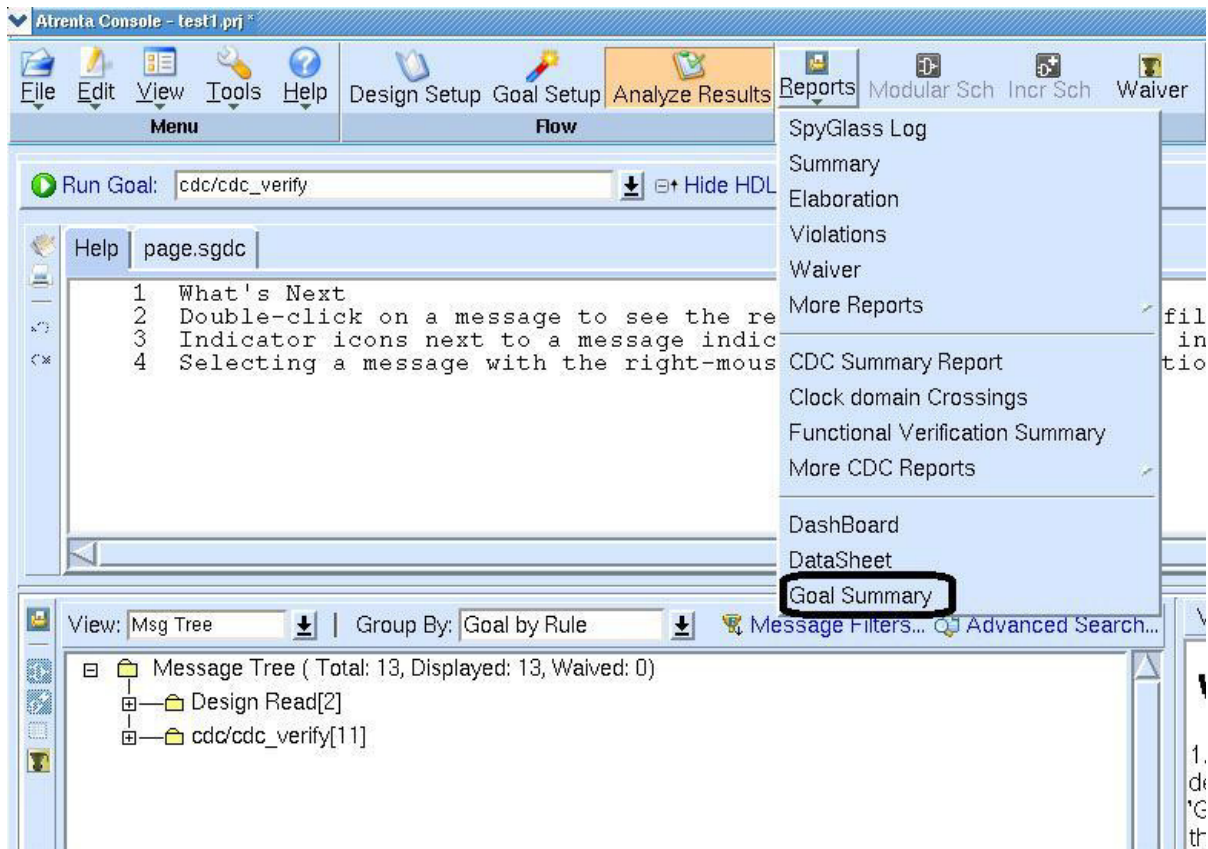


FIGURE 46. Goal Summary Menu

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.

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 Atrenta Console before analyzing other design files.*

- `'include`

NOTE: *Atrenta Console 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 Atrenta Console.

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 Atrenta Console 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 Atrenta Console 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 Atrienta Console project command:

```
define_design_lib L1 -path ./L1_path ' define_library_map
L1 ./L1_path
```

Project File Details

A project file is a Tcl format file that enables easy reading and editing of this file outside the Atrenta Console GUI.

NOTE: *In Atrenta Console, Tcl format is supported only in a project file. However, this format is not supported in Atrenta Console 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.*

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 Atrenta Console GUI, and save the settings into a project file.

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

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

Atrenta Console 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 Atrenta Console 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. However, if you select the *Write unmodified options in Project File* check box in the Properties dialog (see *File > Project Properties* in the Atrenta Console Reference Guide), the design read options that you have not modified are also written in the project file.

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.

For more details on common options, refer to the *Atrenta Console*

Reference Guide.

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.

Value	Description
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.
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: <i><arg-number><arg-type><arg-sort-order></i>
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: <div style="text-align: center;"> <p>Attribute '%1' found on enumerated type '%2' used for encoding FSM states</p> </div> 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, refer to the [Specifying a Current Methodology](#) topic.

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, Atrenta Console 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 (highlighted in blue) 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 (highlighted in blue) 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: Atrenta Console 4.3.0
# Last Updated On Tue Aug 4 19:11:48 2009
#
# -----
```

Project File Details

```
##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 /delsoft/testcases
/DDR/case100
set_option enable_gateslib_autocompile yes

##Goal Setup Section

current_methodology /delsoft/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 /delsoft/
testcases/DDR/case100/Project-1.prj

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

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

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 <code>ff</code> method)	Only single-clock flip-flops are processed for translation. There can be different clocks for master and slave latches as determined by the <code>clocked_on_also</code> attribute in the cell library. Such cells are discarded. If any output function does not involve <code>next_state</code> functional node (inverting or non-inverting internal node), the cell is discarded.
Latches (cells described using Liberty <code>latch</code> method)	Only single enable latches are processed. Dual-enable latch cells are designated by the <code>enable_also</code> attribute and are discarded. If any cell output function does not involve the <code>data_in</code> functional node, then the cell is discarded.

Cell Type	Support Description
Flopbanks and Latchbanks (cells described using Liberty <code>ff_bank</code> and <code>latch_bank</code> 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
checkRTLInst	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

Goals That Do Not Use Default Parameter Value

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_sync reset,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_sync reset,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
checkRTLInst	no	yes

Goals That Do Not Use Default Parameter Value

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_sync reset,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_sync reset,no_convergence_at_enable	mbit_macro
one_cross_per_dest	yes	no

Goals That Do Not Use Default Parameter Value

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
checkRTLInst	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

Goals That Do Not Use Default Parameter Value

Parameter	Default Value	Modified Value
clock_reduce_pessimism	latch_en	all
distributed_fifo	no	yes
cdc_reduce_pessimism	mbit_macro, no_convergence_at_sync reset,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_sync reset,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
checkRTLInst	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

Goals That Do Not Use Default Parameter Value

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_sync reset,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_sync reset,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_consist* PREREQ: initial_rtl/
constraint/sdc_quick_check
!HIDE initial_rtl/constraint/io_delay* PREREQ: initial_rtl/
constraint/clock_consist
!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*
```


List of Topics

About This Book	17
Adding a Logo in the Report Header	440
Adding a Tag	413
Adding Comments in an SGDC File.....	306
Adding Design Files	36
Adding Files in GUI	36
Adding Rules in a Goal	261
Advantage of Specifying a Top-Level Design Unit	212
Advantages of Using Precompiling Libraries.....	130
Analyze Results Stage.....	29
Analyzing Selective Design Hierarchy	225
Archiving and Managing Data Generated After Running Goals	509
Argument Details of the waive Constraint	382
Arguments of the spyencrypt Utility	172
Atrenta Console Flow	28
Automatically Compiling Gate Libraries.....	136
Auto-Migration of Waivers	376
Before You Begin.....	23
Black Boxes.....	459
Browser Compatibility	464
Built-in VHDL Libraries That Do Not Require Any Mapping	140
Capturing Domain Inferring Results	335
Central Design Setup	82
Changing the Default Hierarchy Separator of the SDC2SGDC Constraints	332
Changing the Name of the Report	439
Checking the Inferred Information	202
Checks Performed During the Design Read Process	58
Checks Performed on Stopped Design Units	218
Clock Trees	452
Combinational Cell Support.....	536
Combining Single-Step Precompilation and Top-level Run	542
Common Options Section	525
Comparing Goals	291
Comparing Methodologies.....	284
Comparing Results of Multiple SpyGlass Runs.....	118
Comparison Reported in Batch	120

Compiling HDL Files into a Library	131
Compiling Libraries in Mixed-Language Designs	146
Compiling the Set of Verilog and SystemVerilog Files Separately	167
Compiling Verilog Files Containing SystemVerilog Keywords	167
Conditionally Specifying SGDC Constraints	320
Conditions for Auto-Compilation of Gate Libraries.....	139
Configuring a Methodology.....	244
Configuring Columns.....	51
Conflict Resolution at Block-Level.....	355
Conflict Resolution at Top-Level	353
Congestion	460
Constraints Migrated From Block-Level to Chip-Level	347
Constraints	456
Contents of This Book	18
Controlling the RTL Synthesis Engine.....	208
Converting SDC Attributes into SGDC Commands.....	331
Copying a Methodology	287
Copying and Inheriting Methodologies	287
Copying Goals	256
Creating a Configuration File	437
Creating a Configuration File	469
Creating a Methodology	246
Creating a Migration File	346
Creating a Project File	523
Creating a Sub-Methodology	249
Creating a Waiver File	372
Creating an SGDC File	306
Creating and Modifying a Methodology.....	246
Creating and Modifying a Sub-methodology.....	249
Creating Custom Methodologies	269
Creating Encrypted Library Dump	154
Creating Goal-Based Waiver.....	372
Creating Goals	251
Creating Scenarios.....	112
Creating the Success Criteria File.....	473
Cross-probing from the Msg Tree Page	126
Current Limitation with Mixed-language Designs in SpyGlass	190
Customizing Goals	269
Customizing Report.....	504
Customizing the Report Header	507
Data Import Section.....	524

Debugging Issues in Gate Libraries	148
Default Paths of Aggregated Reports	421
Defining a Logical Library	131
Defining a Scope for Constraints.....	307
Defining and Using Variables.....	315
Defining Variables	315
Deleting a Tag	415
Deleting Goals	256
Design Results	121
Design Setup Stage	28
Design Statistics	458
Details of the DashBoard Report.....	493
Details of the DataSheet Report	449
Details of the waive Constraint.....	386
Details Present in a Goal File.....	231
Determining Parameter Precedence	93
Difference between Ignored and Stopped Design Units	221
Directory Structure Created After Running a Scenario.....	115
Displaying the New Goals Dialog.....	253
Dragging and Dropping Sub-Methodologies and Goals	264
Editing Files.....	47
Editing Source Files	117
Effects of Selected Messages in the Schematic	365
Effects of Waiving Messages	375
Enabling the DesignWare Flow	193
Enabling the Feature.....	207
Enabling the SDC-to-SGDC Translation Feature	331
Enabling/Disabling a Goal	259
Enabling/Disabling Rules of a Parent Goal	280
Encrypting IPs by Using the spyencrypt Utility	171
Encrypting IPs Spread Across a Hierarchical Directory Structure	175
Example of a Tcl-based Project File	534
Example of Using the SG_OPERATING_MODE Variable.....	322
Examples of Instantiating VHDL Design Units in Verilog Modules	187
Examples of Using the waive Constraint	389
Existing Waiver Support in SpyGlass	412
Features of Single Step Precompilation	540
File Generated in GUI.....	32
Files Generated to Support Special Features.....	33
Files/Directories Created in Atrenta Console	30
Files/Directories Generated by Default.....	32

Format of an Order File	299
Generated Hierarchical SGDC File(s)	348
Generating a Precompiled Library	135
Generating Aggregated Project Results	422
Generating Dashboard Report in GUI.....	467
Generating Dashboard Report	464
Generating Hierarchical SGDC File.....	348
Generating SGDC Commands as a Part of Design Read	342
Generating SGDC Commands as a Part of Goal Run.....	342
Generating the DashBoard Report in Batch.....	465
Generating the DashBoard Report through Project File.....	465
Generating the DataSheet Report in Batch	443
Generating the DataSheet Report in GUI.....	435
Generating the Datasheet Report through a Project File	444
Generating the HTML Goal Summary Page	511
Generating the Project Summary Report.....	426
Generating the Report through a Project File	427
Generating the Report through GUI.....	426
Goal Files	230
Goal Setup and Run Stage	28
Goal Setup Section	529
Goal Summary	516
Goals That Do Not Use Default Parameter Value	544
GUI Details	26
GuideWare Reference Methodology	234
Handling Directional Clocks	339
Handling Duplicate Constraint Specifications.....	318
Handling False Paths	338
Handling for-generate Constructs.....	359
Handling Interdependencies between Different Arguments	311
Handling Internal Messages	417
Handling Language Warning Messages	416
Handling Multi-cycle Paths	339
Handling Mutually Exclusive Clocks	339
Handling Nets Declared in a Sequential Block	319
Handling of Generated Clocks.....	336
Handling Out of Memory Situations	205
Handling SpyGlass Built-In Messages	416
Handling SV Structure or Union	358
Handling Syntax Error Messages.....	416
Handling Synthesis Error Messages	417

Handling Synthesis Warning Messages	416
Handling SystemVerilog Interface Containing a Modport	357
Handling SystemVerilog Interface Port/Terminal	357
Handling SystemVerilog Objects in SGDC.....	357
Handling Unsaved Changes in Waiver Files.....	374
Identifying Common Syntax Errors and Issues	59
Identifying Modules	206
Ignoring Files and Design Units From SpyGlass Analysis	220
Ignoring Files Containing Design Units.....	221
Ignoring Files from SpyGlass Analysis	50
Ignoring Individual Design Units.....	222
Ignoring the SpyGlass Waiver Pragmas	411
Impact of the addrules Option While Using Pre-compiled Dump	159
Impact of the Feature	207
Impact of the ignorerules Option While Using Pre-compiled Dump.....	159
Implementing Scoping in SGDC Commands	351
Implications After Stopping Design Units	217
Importing Block-Level SGDC Commands to Chip-Level	346
Importing Goals	255
Including a Waiver File in Another Waiver File	375
Including an SGDC File in Another SGDC File.....	312
Including and Inheriting GuideWare Goals	269
Including HDL Files in the Logical Library	133
Including Product-Specific Data in the Report.....	505
Including/Inheriting Goals in a Goal File	270
Including/Inheriting Goals in the MCS Window.....	277
Incremental Mode Analysis	108
Inferring Black Boxes.....	200
Inferring cdc_false_path for Clocks in Different Domains.....	334
Inferring Language from File Extension During Compilation	161
Inheriting a Methodology.....	288
Instantiating as Component Instance	182
Instantiating as Entity Instance	184
Instantiating Verilog Modules in VHDL Architectures	182
Instantiating VHDL Design Units In Verilog Modules	186
Interpreting Synthesis Pragmas.....	209
Interpreting Synthesis Pragmas.....	210
Introducing Goals.....	21
Introducing Methodologies.....	22
Introducing the Incremental Mode Feature.....	118
Introducing the Use Model for IP Encryption in SpyGlass.....	170

Invoking Atrenta Console Graphical User Interface	24
Invoking Atrenta Console on a 64-bit Machine	25
IO Definitions.....	451
Language-Specific Behavior While Specifying a Top-Level Module.....	215
Library Searching Mechanism	142
Licensing Requirements.....	464
Limitations.....	344
Limiting Analysis of Memories.....	208
Limiting the Number of Messages Generated	369
Limiting the Number of Messages Reported for a Rule.....	369
List of DesignWare Modules Supported in SpyGlass	195
Loading the Previous Session	24
Makefile Based Support in Step Precompilation	541
Managing Datasheet and Dashboard Reports	518
Managing Reports.....	509
Managing the Design Hierarchy	212
Map File	302
Mapping a File Extension with a Compilation Language	160
Mapping between VHDL Generics and Verilog Parameters.....	189
Mapping Data Types	189
Mapping File Extensions.....	45
Merging the Differences.....	286
Messages Affecting Multiple Source Lines/Files.....	367
Methodology Used by Atrenta Console	22
Migrating Custom Goals.....	291
Migrating Goals	297
Modifying a Goal.....	67
Modifying a Methodology	248
Modifying a Sub-Methodology.....	251
Modifying a Tag.....	414
Modifying and/or Deleting Scenarios.....	114
Modifying Goal Properties	257
Modifying Goals.....	256
Modifying Parameters of a Goal	263
Module Dashboard	500
Multiple Lines Affected in Different Source Files	368
Multiple Lines Affected in the Same Source File.....	368
Multiple Messages Selected	368
Multiple Top-Level Design Units	214
Naming and Mapping Verilog Libraries	141
Naming Convention of a Goal File	230

Order File	298
Overview.....	129
Overview.....	21
Overview.....	229
Overview.....	303
Overview.....	35
Overview.....	363
Overview.....	419
Parsing SGDC Files	327
Performing Hierarchical Rule-Checking in 'celldefine Modules	227
Performing Rule-Checking on 'celldefine Modules	226
Performing Sanity Checks for Parameters	101
Performing Syntax Checking in SGDC Files.....	327
Performing Version Control	46
Power Clocks	455
Power Results	125
Power	454
Precompiling Multiple Libraries in a Single SpyGlass Run	539
Precompiling Verilog Libraries	140
Prerequisites for Enabling DesignWare Flow	192
Preserving all instances and nets in a design	209
Processing of SGDC Files	327
Processing SpyGlass Design and Waiver Pragmas	328
Project Current Working Directory	31
Project File Details.....	523
Project File	30
Project Summary Report	426
Project Working Directory	30
Rearranging HDL Files.....	46
Recognizing Clocks	330
Recommended Goals for Generating DataSheet Report.....	446
Reporting Messages at Module Boundary	206
Reset Trees	453
Returning Back to the Goal Setup & Run Stage	118
Reusing Netlist of DesignWare Modules during SpyGlass Analysis	194
Re-using Simulation Scripts	521
Running Custom Goals	72
Running Design Read in Batch.....	58
Running Design Read in GUI	55
Running Design Read.....	55
Running Goals in Parallel	72

Running Prerequisite Goals	109
Running Scenarios	114
Sample Order File	555
Saving the Generated SGDC Commands in a File	341
Scoping When Design is at the Block-Level.....	354
Scoping When Design is at Top-Level	352
Searching for Input Files.....	420
Searching Instances.....	62
Searching Master Instance in Mixed-Language Mode	147
Selecting a Custom Methodology	281
Selecting a Goal	65
Selecting Auxiliary Messages without Selecting a Main Message	367
Selecting Non-Static Auxiliary Messages	366
Selecting Static Auxiliary Messages	366
Selection of Goal Files based on Language Mode	232
Sequential Cell Support	536
Setting a Top-Level Design Unit.....	213
Setting Default Waiver File.....	373
Setting Parameters and Constraints for Selected Goal.....	93
Setting Stop Files	50
Setting Up the Goal in Batch Mode	109
Setting up the Goal.....	93
SGDC Convention for Packed Arrays.....	309
SoC Dashboard	493
Specifying a Cache Directory	139
Specifying a Current Methodology	242
Specifying a List of .sglib Files	44
Specifying a Reference Environment Variable	289
Specifying a Top-level Design Unit	212
Specifying an Active Methodology	238
Specifying an Additional Path	289
Specifying Compilation Options in a Source File	165
Specifying Compressed Verilog Designs	44
Specifying Configuration Name with current_design Command	310
Specifying Details in the New Goal Dialog	253
Specifying Encrypted Files for SpyGlass Analysis	177
Specifying Encrypted Files through a Project File	180
Specifying Encrypted Files through GUI	177
Specifying Files in the Order of Their Dependencies	166
Specifying Functionality Information of Gate Cells	42
Specifying Modes in Which Libraries Should be Compiled.....	130

Specifying Multiple current_design Specifications for a Design Unit.....	309
Specifying Multiple Technology Libraries of the Same Name	150
Specifying Multiple Values for a Constraint Argument	311
Specifying Optional Environment Variables.....	23
Specifying Path of DesignCompiler Installation	192
Specifying Pragmas in HDL Code	199
Specifying Precompiled Libraries for SpyGlass Analysis	149
Specifying SGDC Files to SpyGlass.....	305
Specifying Signal Names based on Design Hierarchy	313
Specifying Signal Names based on Signal Types	313
Specifying Signal Names	313
Specifying the Mode of an SDC File	341
Specifying the Mode of Domain Inference	333
Specifying Verilog Libraries by Using the 'uselib Statement	146
SpyGlass CDC Solution Results.....	122
SpyGlass Constraints Solution Results	122
SpyGlass DFT Solution Results	124
SpyGlass TXV Solution Results	122
Stage 1: Setting up the Design (Design Setup).....	36
Stage 2: Selecting a Goal (Goal Setup & Run)	64
Stage 3: Analyzing a Design (Analyze Results)	116
Starting a New Session	24
Stopping Black Box Analysis	204
Stopping Design Units.....	216
Structure of a Project File	523
Structure of Precompiled Verilog Libraries.....	142
Structure of the GuideWare Reference Methodology	234
Support for Hierarchical Waivers.....	400
Support for Virtual Clocks in sdc2sgdc Flow.....	343
Supported HDL Directives.....	519
Supported Library Cells	536
Supported Pragmas for Verilog	199
Supported Pragmas for VHDL.....	199
Switching to the Old Dashboard Report	515
Syntax of the waive Constraint.....	382
Tagging Messages	413
Tcl Format Support in the Configuration File.....	440
Tcl Format Support in the Configuration File.....	471
Testability	457
The DashBoard Report	462
The DataSheet Report.....	433

The Methodology Configuration System	109
Timing	459
Tips for Debugging Syntax Errors.....	60
Translating set_clock_sense command	340
Translating set_disable_timing command.....	340
Translating set_mode command	341
Typographical Conventions	19
Understanding Different Flows for Using This Feature	342
Understanding the Black Box Inference Feature	201
Updating Rules of a Goal	259
Using DesignWare Functions	198
Using Encrypted Library Dump	155
Using File Extension Based Compilation Flow	168
Using Intermediate Logical Library Name Support in VHDL	151
Using Regular Expressions and Wildcard Characters	390
Using the active_methodology Option.....	241
Using the AUTOENABLE_GATESLIB_AUTOCOMPILE Key	138
Using the Black Box Inference Feature.....	202
Using the Corrected Inferred Information.....	204
Using the Dual Design Read (DDR) Flow	101
Using the enable_gateslib_autocompile Option	138
Using the force_gateslib_autocompile Option.....	138
Using the GUI to Automatically Compile Libraries	137
Using the Incremental Mode Feature	119
Using the Results Pane to Waive Messages.....	379
Using the SG_OPERATING_MODE Variable	321
Using the Top and Stop Features Together	218
Using the Waiver Editor Window	377
Using Variables	315
Using Verilog Constructs.....	54
Validating the Generated Hierarchical SGDC File	349
Verilog Modules Instantiated in VHDL Design Units.....	147
VHDL Library Design Units Instantiated in Verilog Modules	147
Viewing and Adding Options for an Included or Inherited Goal	278
Viewing and Changing Design Read Options	52
Viewing Built-In Messages for Precompiled Libraries	156
Viewing CSV Reports.....	432
Viewing Different Type of Results.....	120
Viewing Directories Created After Goal Run	81
Viewing Encryption Summary in a Report.....	176
Viewing Goal Summary	117

Viewing Include Files	51
Viewing Messages after Running Design Read	59
Viewing Order of Goals Defined in an Order File	299
Viewing Reports	60
Viewing Results of Different Scenarios and Goals	126
Viewing Rules and Parameters of Included/Inherited Goals.....	279
Viewing Source Files	62
Viewing the DashBoard Report	492
Viewing the DataSheet Report.....	446
Viewing the HTML Report for Comparison	296
Viewing the HTML Report.....	427
Viewing the Project Summary Report	427
Virtual to Real Clock Mapping	344
Waiver File	372
Waiving Messages by File	51
Waiving Messages by Using SpyGlass Pragmas	402
Waiving Messages by Using the waive Constraint.....	381
Waiving Messages in Waiver/SGDC Files	411
Waiving Messages through a Project File.....	381
Waiving Messages through GUI	377
Waiving Messages	371
Waiving Rule Messages for a Block of Code	404
Waiving Rule Messages for a Single Line of Code	406
Wildcard Support at Block-Level	355
Wildcard Support at Top-Level	353
Working with 'celldefine Modules	226
Working with Black Boxes.....	200
Working with Compressed Gate Library Files	153
Working with DesignWare® Modules.....	192
Working with Encrypted Compiled Libraries	154
Working with Encrypted Design Files	170
Working with Mixed-Language Designs.....	182
Working with Multiple Messages	365
Working with Precompiled Libraries.....	130
Working with Precompiled Verilog Libraries in Mixed Language Mode	143
Working with Scenarios	111

