

Leda

User Guide

Version 2006.06
June 2006



Comments?
E-mail your comments about this manual to
leda-support@synopsys.com.

SYNOPSYS[®]

Copyright Notice and Proprietary Information

Copyright © 2005 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

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.

Registered Trademarks (®)

Synopsys, AMPS, Arcadia, C Level Design, C2HDL, C2V, C2VHDL, Cadabra, Calaveras Algorithm, CATS, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSPICE, Hypermodel, iN-Phase, in-Sync, Leda, MAST, Meta, Meta-Software, ModelAccess, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PowerMill, PrimeTime, RailMill, Raphael, RapidScript, Saber, SiVL, SNUG, SolvNet, Stream Driven Simulator, Superlog, System Compiler, Testify, TetraMAX, TimeMill, TMA, VCS, Vera, and Virtual Stepper are registered trademarks of Synopsys, Inc.

Trademarks (™)

abraCAD, abraMAP, Active Parasitics, AFGen, Apollo, Apollo II, Apollo-DPII, Apollo-GA, ApolloGAI, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BCView, Behavioral Compiler, BOA, BRT, Cedar, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, Cyclelink, Davinci, DC Expert, DC Expert Plus, DC Professional, DC Ultra, DC Ultra Plus, Design Advisor, Design Analyzer, Design Vision, DesignerHDL, DesignTime, DFM-Workbench, DFT Compiler, Direct RTL, Direct Silicon Access, Discovery, DW8051, DWPCI, Dynamic-Macromodeling, Dynamic Model Switcher, ECL Compiler, ECO Compiler, EDANavigator, Encore, Encore PQ, Evaccess, ExpressModel, Floorplan Manager, Formal Model Checker, FoundryModel, FPGA Compiler II, FPGA Express, Frame Compiler, Galaxy, Gattran, HDL Advisor, HDL Compiler, Hercules, Hercules-Explorer, Hercules-II, Hierarchical Optimization Technology, High Performance Option, HotPlace, HSPICE-Link, iN-Tandem, Integrator, Interactive Waveform Viewer, i-Virtual Stepper, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, JvXtreme, Liberty, Libra-Passport, Library Compiler, Libra-Visa, Magellan, Mars, Mars-Rail, Mars-Xtalk, Medici, Metacapture, Metacircuit, Metamanager, Metamixsim, Milkyway, ModelSource, Module Compiler, MS-3200, MS-3400, Nova Product Family, Nova-ExploreRTL, Nova-Trans, Nova-VeriLint, Nova-VHDLint, Optimum Silicon, Orion_ec, Parasitic View, Passport, Planet, Planet-PL, Planet-RTL, Polaris, Polaris-CBS, Polaris-MT, Power Compiler, PowerCODE, PowerGate, ProFPGA, ProGen, Prospector, Protocol Compiler, PSMGen, Raphael-NES, RoadRunner, RTL Analyzer, Saturn, ScanBand, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, Silicon Blueprint, Silicon Early Access, SinglePass-SoC, Smart Extraction, SmartLicense, SmartModel Library, Softwire, Source-Level Design, Star, Star-DC, Star-MS, Star-MTB, Star-Power, Star-Rail, Star-RC, Star-RCXT, Star-Sim, Star-SimXT, Star-Time, Star-XP, SWIFT, Taurus, Taurus-Device, Taurus-Layout, Taurus-Lithography, Taurus-Process, Taurus-Topography, Taurus-Visual, Taurus-Workbench, TimeSlice, TimeTracker, Timing Annotator, TopoPlace, TopoRoute, Trace-On-Demand, True-Hspice, TSUPREM-4, TymeWare, VCS Express, VCSi, Venus, Verification Portal, VFormal, VHDL Compiler, VHDL System Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.

ARM and AMBA are registered trademarks of ARM Limited.

All other product or company names may be trademarks of their respective owners.

Contents

Preface	19
About This Manual	19
Related Documents	19
Manual Overview	19
Typographical and Symbol Conventions	21
Getting Leda Help	22
The Synopsys Web Site	22
Chapter 1	
Leda Overview	23
Introduction	23
What is Leda?	23
How Leda Works	25
Leda Terminology	26
Types of Leda Rules	27
Approaches to Using Leda	29
Using Leda in Batch, GUI, and Tcl Shell Modes	30
Invoking Leda	31
Switching Modes	31
Creating Projects	32
Opening Projects	32
Enabling Design Query Commands	32
Configuring the GUI	33
Typical Leda Usage Scenarios	34
About Design Rules	39
Using .db Files for Checks	39
Limitation with Gates in .db Files	41
About Hardware-Based Rules	42
Finite State Machine Rules	42
Hardware Inference	43
Set and Reset Detection in VHDL and Verilog	49
Rules Leda Cannot Check	50
Chapter 2	
Writing and Checking HDL Designs	51
Introduction	51

Writing & Checking VHDL Designs	52
VHDL Semantic Exceptions	52
Writing & Checking Verilog Designs	56
Verilog Semantic Exceptions	57
Writing & Checking Mixed-Language Designs	59
Instantiating a Verilog Module in a VHDL Architecture	59
Instantiating a VHDL Design Entity in a Verilog Module	59
Mapping Data Types	60
VHDL and Verilog Identifiers	63
Verilog 2001 Support	65
SystemVerilog Support	65
Clock Grouping Feature	66
Netlist Reader	68
Invoking the Netlist Reader	68
Netlist Reader BNF	69
Chapter 3	
Modifying and Creating Rules	71
Introduction	71
About Rules, Rulesets, and Policies	72
Using Configurations	72
Configuring the Rule Wizard	73
Saving Configurations	73
Restoring Configurations	73
Rule Configuration Search Path	74
Global Checking with the Same Rule Configuration	74
Configuring Prepackaged Rules	74
Locking the Rule Wizard	75
Using the Rule Wizard to Configure Rules	77
Policy and Topic Views	77
Configuring Rule Properties	78
Creating New Rules	79
Copying and Modifying Prepackaged Coding Rules	80
Writing New Rules from Scratch	80
Creating New Ruleset Files	81
Creating New Policies	81
Defining Macro Values for Rules	82
Using Predefined Macros to Constrain Identifiers	84
Advanced Macro Programming	85
Constraining Max/Min Attributes to Predefined Values	85

Exporting and Importing Policies	86
Chapter 4	
Checking Designs For Errors	89
Introduction	89
Invoking the Checker GUI	90
Creating Projects to Check HDL Code	91
Propagating Constants	96
Constant Propagation Limitations	98
Using the Rule Wizard to Select or Deselect Rules	98
Using Prebuilt Configurations	99
Policy and Topic Views	100
Selecting or Deselecting Rules	100
Disabling Redundant Rules	101
Deactivating Rules	101
Deactivating Rules with a Rule Configuration File	102
Deactivating Rules from within HDL Source Files	103
Deactivating Verilint Policy Rules	104
Deactivating Rules from the Error Viewer	105
Deactivating Rules By File	106
Translating .leda_select Files	106
Setting & Saving Checker Preferences	108
Running the Checker	109
Top Unit Tab	109
Test Clock/Reset Tab	110
Checkers Tab	111
Fixing Errors Found by the Checker	112
Reviewing Log, History, Errors/Warnings Tab in GUI	114
Displaying Error Messages for STARC Policies	115
Getting Prepackaged Rule Help for STARC Policies	115
Sorting the Error Viewer Display	116
Filtering the Error Viewer Display	117
Error Report Displays	117
Viewing the Design Report	120
Using the Path Viewer	121
Using Trace Forward and Trace Backward	123
Using the Clock and Reset Tree Browsers	126
Saving Error Reports	127
Post-processing Batch Mode Log Files	128
Generating Leda Summary Information (Info Report)	129

Updating Projects	130
Chapter 5	
Using the SDC Checker	133
Introduction	133
Leda Quality Checks	133
Top-versus-Block SDC Checks	133
SDC Equivalency Checks	134
Simplified Usage Model for SDC Checker	135
Supported SDC File Tcl Commands	137
Specifying Design Objects	139
Handling Errors in SDC Files	140
Leda SDC Checker Tcl Commands	140
Using a Tcl File For SDC Checks	141
Defining Parameters for SDC Rules	142
Chapter 6	
Using Leda Batch Mode	145
Introduction	145
Basic Usage Models and Rule Types	145
Configuring the Checker	146
Using plibs to Set Library Logical/Physical Mapping	146
Running Leda in Batch Mode	148
Common Command-Line Options and Switches	148
VHDL Command-Line Options	157
Verilog Command-Line Options	159
Leda Batch Example Invocations	162
Generating Log Files in Batch Mode	163
Generating Projects in Batch Mode	163
Verilog-only Projects	163
VHDL-only Projects	164
Mixed-Language Projects	165
Checker Batch Mode Results	166
Checker Return Status	166
Viewing Checker Results	167
Checking the Environment	167
Chapter 7	
Using Leda GUI Mode	169
Introduction	169
Invoking the Checker/Specifier GUI	170

Checking Your Environment	171
Selecting a Text Editor	172
The File Menu	173
The Project Menu	175
The Check Menu	176
The Report Menu	177
The View Menu	178
The Window Menu	178
The Help Menu	178
Managing Source Files From the GUI	181
Using Pop-up Menus in the Files Tab	182
Managing Library Units From the GUI	184
Using Pop-up Menus in the Modules/Units Tab	185
Generating Log Files in GUI Mode	186
Chapter 8	
Using Leda Tcl Shell Mode	187
Introduction	187
Invoking Leda in Tcl shell Mode	187
Enabling Netlist Checks	188
Changing Leda Modes	188
Sourcing a Tcl Script in Leda	188
Built-in Tcl Commands	189
Getting Help on Leda Tcl Commands	189
Collections	190
Current Limitation	192
Regular Expressions	192
Using Regular Expressions with Hierarchy	193
Anchoring Regular Expressions	193
Using Regular Expressions with Busses	194
Filter Expressions	194
Using the -filter Option	195
Rule Tcl Command Reference	197
is_64bit	197
add_to_collection	197
all_clocks	197
all_inputs	198
all_instances	198
all_outputs	198
all_registers	198

append_to_collection	198
create_operating_conditions	199
compare_collections	199
connect_power_domain	200
copy_collections	200
create_power_domain	200
create_power_net_info	201
delete_operating_conditions	201
disable_isolation_cell_recognition	202
enable_isolation_cell_recognition	202
filter_collection	202
foreach_in_collection	203
get_all_input_boundaries_from_power_domain	203
get_all_output_boundaries_from_power_domain	203
get_cells	203
get_clocks	204
get_nets	204
get_nth_power_net	205
get_object_name	205
get_power_cells	205
get_power_down	206
get_power_down_ack	206
get_power_net_max_voltage	206
get_power_net_min_voltage	206
get_power_net_source_port	207
get_power_net_type	207
getn_power_net	207
get_pins	207
get_ports	208
get_power_domains	208
infer_power_domain	209
infer_power_domains	209
index_collection	209
print_config_summary	210
query_objects	210
remove_from_collection	210
remove_isolation_cell	211
remove_level_shifter	211
remove_power_domain	211
remove_power_net_info	212

report_clock_gating_cells	212
report_enable_pin	213
report_isolation_cells	213
report_level_shifter	213
report_operating_conditions	214
report_pin_voltages	214
report_power_domain	214
report_power_net_info	215
report_power_pins	215
report_power_switches	215
reset_isolation_cell_recognition	216
rule_deselect	216
rule_get_parameter	217
rule_get_selection	218
rule_get_all_masters_from_topic	219
rule_get_all_rules_from_master_id	220
rule_get_all_topics	221
rule_get_configuration	222
rule_get_current_configuration	223
rule_get_policies	224
rule_get_policy_attributes	225
rule_get_predefined_configurations	226
rule_get_rules	227
rule_get_ruleset_attributes	228
rule_get_rulesets	229
rule_get_templateset_attributes	230
rule_get_templatesets	231
rule_link	232
rule_load	232
rule_load_configuration	233
rule_manage_policy	234
rule_patch	235
rule_save_configuration	235
rule_get_current_configuration	236
rule_set_default_configuration	237
rule_set_predefined_configuration	238
rule_select	239
rule_set_html	240
rule_set_message	241
rule_set_parameter	241

rule_set_severity	247
set_clock_gating_cell	247
set_enable_pin	248
set_level_shifter	248
set_operating_conditions	248
set_pin_voltage	249
set_power_pin	249
set_power_domain	250
set_power_domain_ctrl	250
set_power_off_value	250
set_power_switch	251
sizeof_collection	251
sort_collection	251
Project Tcl Command Reference	253
project_add_library	253
project_build	254
project_delete	255
project_get_all_files	255
project_get_file_attributes	256
project_get_library_attribute	257
project_get_option_attribute	258
project_get_ports	258
project_get_top_units	259
project_get_unit_kinds_from_library	259
project_get_units_from_file	260
project_get_units_from_library	261
project_get_working_libraries	262
project_new	262
project_open	263
project_quit	263
project_read	264
project_record_cmd	264
project_remove_file	265
project_remove_library	265
project_save	266
project_specify_files	266
project_specify_libraries	267
project_specify_name	268
project_specify_options	269
project_update	269

Checker Tcl Command Reference	271
check	271
checker_get_design_constraints	276
checker_get_options	276
checker_set_design_constraints	278
checker_set_options	280
current_design	282
elaborate	283
link	285
propagate	286
read_constraints	287
read_files	289
read_sverilog	293
read_verilog	296
read_vhdl	299
report	300
run	301
sdc_apply	303
set_case_analysis	304
verify	305
Generating Log Files in Tcl Mode	307
Reserved Variables	307
Appendix A	
Managing VHDL Libraries and Files	313
Introduction	313
Setting Libraries	313
Setting Resource Libraries	314
Building Libraries	314
Adding Files to VHDL Resource Projects	315
Adding Libraries to VHDL Resource Projects	315
Creating Local VHDL Resource Libraries	316
Appendix B	
Leda Environment Variables	317
Introduction	317
Setting Leda Environment Variables	317
Using Leda Environment Variables	318

Appendix C

Leda Prebuilt Configurations	321
Overview	321
RTL Prebuilt Configuration	322
Gate-level Prebuilt Configuration	325
Leda-classic Prebuilt Configuration	327
CDC Prebuilt Configuration	387
SDC-postlayout Prebuilt Configuration	388
SDC-prelayout Prebuilt Configuration	390
SDC-RTL Prebuilt Configuration	393
SDC-top-versus-block Prebuilt Configuration	396
SDC-equivalency Prebuilt Configuration	397

Appendix D

Leda Duplicated Rules	399
Introduction	399
Disabling Redundant Rules	399
Duplicated Rule List	400

Appendix E

Errors and Warnings Message List	433
Introduction	433
Verilog Compilation Warnings	433
Verilog Compilation Failures	437
Deselectable Messages	440
Elaboration Failure Messages	442
Elaboration Error Messages	442
Elaboration Warning Messages	444
Elaboration Note Messages	446
Index	447

Figures

Figure 1:	Leda Rule Specifier and Checker Overview	24
Figure 2:	Approaches to Using Leda	29
Figure 3:	Leda Modes of Operation	30
Figure 4:	Signal CLK is a primary clock	43
Figure 5:	Signal CLK is a primary clock	43
Figure 6:	Signal CLK is also a primary clock	44
Figure 7:	Signal D1 is a generated clock from primary clock CLK	44
Figure 8:	INT1 is a generated clock as no connection to a primary port	45
Figure 9:	INT1 is a generated clock due to disconnection	45
Figure 10:	Gated clock	46
Figure 11:	Gated clock	47
Figure 12:	Checker Control Panel	75
Figure 13:	Locked Rule Wizard Warning	76
Figure 14:	Rule Wizard Window	77
Figure 15:	Invoking the Policy Manager	81
Figure 16:	Leda Checker Main Window	90
Figure 17:	Project Creation Wizard Window	91
Figure 18:	Leda Checker Results	95
Figure 19:	Constant Propagation for Test Mode	96
Figure 20:	Rule Wizard Window	98
Figure 21:	Deactivating Rules from Error Viewer	105
Figure 22:	Deselect Rules by File in Rule Wizard	106
Figure 23:	Checker Options in Application Preferences	108
Figure 24:	Specify Design Information Window (Top Units Tab)	109
Figure 25:	Test Clock/Reset Tab	110
Figure 26:	Checkers Tab	111
Figure 27:	Checker After Check	112
Figure 28:	Log, History, Error/Warnings Tab	114
Figure 29:	Error Viewer Preferences Window	116
Figure 30:	Error Viewer Summary	116
Figure 31:	Severity, Message, and Label in Rule Display	117
Figure 32:	File Level in Rule Display	118
Figure 33:	HDL Fragments in Rule Display	118
Figure 34:	Error Level File Display	119
Figure 35:	HDL Fragments in File Display	119
Figure 36:	Leda Design Report	120

Figure 37: Invoking the Path Viewer	121
Figure 38: Path Viewer Window	121
Figure 39: Hierarchy Browser Window	122
Figure 40: Hierarchy Types in Path Viewer	122
Figure 41: Traceable Objects in Path Viewer	123
Figure 42: Extended/Standalone Path Viewer Window	124
Figure 43: Clock View in Clock and Reset Tree Browser	126
Figure 44: Project Update Wizard	130
Figure 45: Tcl File with SDC Checker Commands	141
Figure 46: Leda Checker Main Window	170
Figure 47: Leda Info Report Tab Display	171
Figure 48: Set Text Editor Window	172
Figure 49: Source File Manager Window	181
Figure 50: Library Unit Manager Window	184

Tables

Table 1:	Documentation Conventions	21
Table 2:	Key Terminology in Leda	26
Table 3:	Types of Leda Rules	27
Table 4:	VHDL Design Entity Instantiations in Verilog Modules	59
Table 5:	Mapping VHDL Ports to Verilog Ports	60
Table 6:	Mapping Verilog Ports to VHDL Ports	61
Table 7:	Mapping VHDL bit Types to Verilog States	61
Table 8:	Mapping VHDL std_logic Types to Verilog States	61
Table 9:	Mapping Verilog States to VHDL std_logic and bit Types	62
Table 10:	Mapping VHDL Identifiers to Verilog Identifiers	64
Table 11:	Environment Variables in Clock Grouping Feature	66
Table 12:	Rule Severity in Rule Wizard	78
Table 13:	Choosing a Method for Creating New Rules	79
Table 14:	Command-Line Checker Error Report Options	129
Table 15:	Supported SDC Design Constraint Commands	137
Table 16:	Supported SDC Design Object Commands	139
Table 17:	Common Command-Line Options and Switches	148
Table 18:	VHDL Command-Line Options and Switches	157
Table 19:	Verilog Command-line Options and Switches	159
Table 20:	Checker Return Status	166
Table 21:	File Menu Choices	173
Table 22:	Project Menu Choices	175
Table 23:	Check Menu Choices	176
Table 24:	Report Menu Choices	177
Table 25:	Help Menu Choices	178
Table 26:	Source File Levels in Display	181
Table 27:	Project Pop-up Menu Choices	182
Table 28:	Library Pop-up Menu Choices	182
Table 29:	Source File Pop-up Menu	183
Table 30:	Unit Pop-up Menu	183
Table 31:	Library Unit Levels in Display	184
Table 32:	Project Pop-up Menu Choices	185
Table 33:	Library Pop-up Menu Choices	186
Table 34:	Attributes of Objects supported by Collection	190
Table 35:	192
Table 36:	Leda Environment Variables	318

Table 37: RTL Prebuilt Configuration	322
Table 38: Gate-level Prebuilt Configuration	325
Table 39: Leda-classic Prebuilt Configuration	327
Table 40: CDC Prebuilt Configuration	387
Table 41: SDC-postlayout Prebuilt Configuration	388
Table 42: SDC-prelayout Prebuilt Configuration	390
Table 43: SDC-RTL Prebuilt Configuration	393
Table 44: SDC-top-versus-block Prebuilt Configuration	396
Table 45: SDC-equivalency Prebuilt Configuration	397
Table 46: Duplicated Rule List	400

Preface

About This Manual

This manual is designed for engineers who want to write rules using the Leda Specifier tool or check HDL source files against different sets of rules using the Leda Checker tool. If you want to write new coding rules, and are unfamiliar with VerSL and VRSL, you should read and work the examples in the [Leda Rule Specifier Tutorial](#) before using this book. Similarly, if you want to write design rules that run against your elaborated design database, see the [Leda Tcl Interface Guide](#) or [Leda C Interface Guide](#). This manual is intended for use by design and quality assurance engineers who are already familiar with Tcl and VHDL or Verilog.

Related Documents

This manual is part of the Leda documentation set. To see a complete listing, refer to the [Leda Document Navigator](#).

Manual Overview

This manual contains the following chapters and appendixes:

Preface	Describes the manual and lists the typographical conventions and symbols used. Explains how to get technical assistance.
Chapter 1 Leda Overview	An overview of Leda, including a diagram of how the tool works, explanations about the different types of Leda rules, definitions of key terms, different modes of operation (GUI, batch, and Tcl shell), and the recommended approaches for using Leda to complete different verification tasks.

Chapter 2 Writing and Checking HDL Designs	Shows recommended organizations for VHDL, Verilog, and mixed-language projects. Explains syntax to use when instantiating across languages, and the data type mappings for VHDL and Verilog.
Chapter 3 Modifying and Creating Rules	Explains how to configure prepackaged rules, copy-and-modify prepackaged rules, and write new rules from scratch using the Specifier tool.
Chapter 4 Checking Designs For Errors	Explains how to organize HDL source files into projects, check these HDL files against rules that you select, and fix any errors that are found.
Chapter 5 Using the SDC Checker	Explains how to use the Synopsys Design Constraint (SDC) checker tool to check SDC files for consistency and correctness, both internally and with respect to the design.
Chapter 6 Using Leda Batch Mode	Explains how to use the Leda Checker tool in batch mode from the command line. Includes complete syntax for all command-line options and switches, as well as Verilog-only and VHDL-only options and switches.
Chapter 7 Using Leda GUI Mode	Provides information on how to use the Leda GUI, including descriptions of all the menus available from the Specifier and Checker main windows.
Chapter 8 Using Leda Tcl Shell Mode	Provides reference and syntax information for the custom Tcl procedures available in Leda for managing rules, projects, and Checker runs.
Appendix A Managing VHDL Libraries and Files	Provides detailed information on how to set up and manage VHDL resource libraries and files. “Setting Libraries” on page 313
Appendix B Leda Environment Variables	Lists all of the Leda environment variables and their uses.
Appendix C Leda Prebuilt Configurations	Lists of all the rules contained in the major prebuilt configurations.
Appendix D Leda Duplicated Rules	Lists of all the redundant rules.
Appendix E Errors and Warnings Message List	Lists of all the errors, warnings, fatal, and note messages.

Typographical and Symbol Conventions

Table 1 describes the typographical conventions used in this manual.

Table 1: Documentation Conventions

Convention	Description and Example
%	Represents the UNIX prompt.
Bold	User input (text entered by the user). % cd \$LMC_HOME/hd1
Monospace	System-generated text (prompts, messages, files, reports). No Mismatches: 66 Vectors processed: 66 Possible"
<i>Italic or Italic</i>	Variables for which you supply a specific value. As a command line example: % setenv LMC_HOME <i>prod_dir</i> In body text: In the previous example, <i>prod_dir</i> is the directory where your product must be installed.
(Vertical rule)	Choice among alternatives, as in the following syntax example: -effort_level low medium high
[] (Square brackets)	Enclose optional parameters: <i>pin1</i> [<i>pin2</i> ... <i>pinN</i>] In this example, you must enter at least one pin name (<i>pin1</i>), but others are optional ([<i>pin2</i> ... <i>pinN</i>]).
TopMenu > SubMenu	Pull-down menu paths, such as: File > Save As ...

Getting Leda Help

For help with Leda, send a detailed explanation of the problem, including contact information, to leda-support@synopsys.com.

The Synopsys Web Site

General information about Synopsys and its products is available at this URL:

<http://www.synopsys.com>

1

Leda Overview

Introduction

This chapter provides an overview of Leda, in the following major sections:

- [“What is Leda?” on page 23](#)
- [“How Leda Works” on page 25](#)
- [“Leda Terminology” on page 26](#)
- [“Types of Leda Rules” on page 27](#)
- [“Approaches to Using Leda” on page 29](#)
- [“Using Leda in Batch, GUI, and Tcl Shell Modes” on page 30](#)
- [“About Design Rules” on page 39](#)
- [“Using .db Files for Checks” on page 39](#)
- [“About Hardware-Based Rules” on page 42](#)
- [“Rules Leda Cannot Check” on page 50](#)

What is Leda?

Leda is a system-to-netlist Checker tool that comes with prepackaged rules to check your Verilog or VHDL designs against various coding standards and design rules.

Leda contains an optional Specifier tool that you can use to define your own coding rules using the supplied VerSL and VRSL macro-based rule programming languages for Verilog and VHDL. Leda also features complete APIs that you can use to develop netlist rules in Tcl or C to run against the elaborated design database.

After you elaborate your design in the Leda environment, you can use the built-in Tcl shell and a set of predefined procedures to run interactive queries on your design. With full support for Verilog 2001 and System Verilog 3.0 (except assertions), combined with extensive RTL and netlist checks, Leda can check your designs from top to bottom for errors that may cause problems in the downstream simulation, synthesis, and equivalence checking flows. For example, Leda can check for synthesizability, simulatability, portability, and optimal performance.

The best way to learn Leda is to test one of your designs with the Checker and the prepackaged policies (sets of rules). The prepackaged rules are designed to meet most hardware design needs. You can also use the Specifier or the Tcl/C APIs to create your own custom rules or configure prepackaged rules to meet your own design team's needs, as shown in Figure 1.

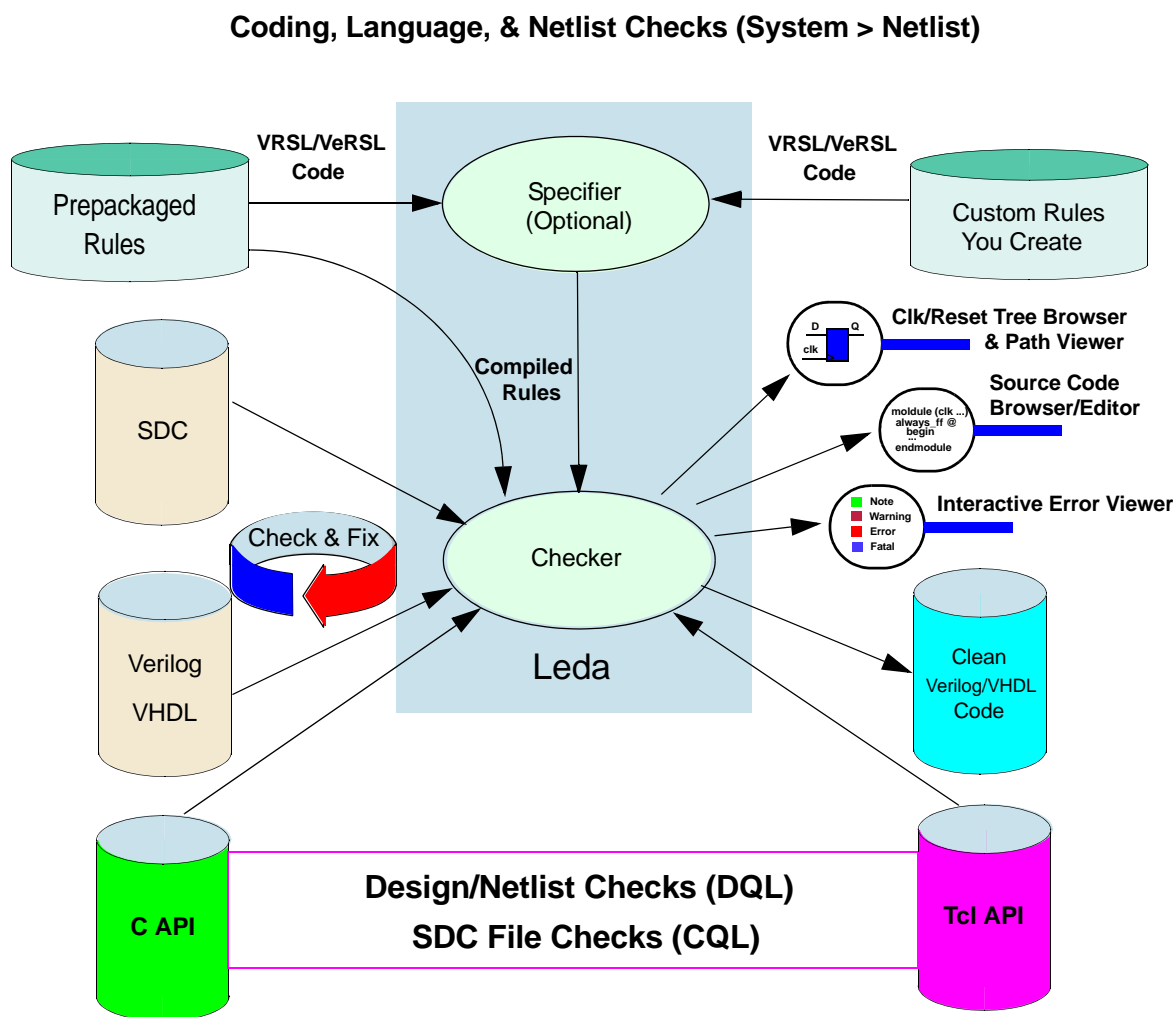


Figure 1: Leda Rule Specifier and Checker Overview

How Leda Works

For block-level or coding rules, if there isn't a prepackaged rule available that meets your needs, you use the Specifier to define templates and rules that jointly describe exactly what your input VHDL or Verilog code should look like. Templates are models of how the code should appear (for example, what HDL constructs should or should not be present, in what order, and so forth).

You select templates and attributes from predefined sets, and program constraints using VRSL/VerSL commands (see [“Approaches to Using Leda” on page 29](#)). You then use the Specifier to compile the rule source code into object files. Note that you need a Specifier license to build and compile your own rules.

For design or netlist rules, if there isn't a prepackaged rule available that meets your needs, you use the Tcl or C APIs to develop rules that run against the elaborated design database. You then integrate your compiled rules (C) or TCL scripts (.tcl) into the Leda environment using VerSL wrappers. Once integrated this way, you use the rules just like other prepackaged or user-defined rules. For more information, see the [Leda Tcl Interface Guide](#) or [Leda C Interface Guide](#).

You can run checks from the GUI, in batch mode from the command line, or using the Tcl shell (see [“Using Leda in Batch, GUI, and Tcl Shell Modes” on page 30](#)). You use the Checker, which is built into the Specifier (or purchased standalone), to designate Verilog or VHDL input files that you want to compare against coding rules that you select with the click of a mouse. The Leda Checker analyzes your Verilog and VHDL source code, and produces error messages indicating which lines in the code violate the rules. You can then:

- Visualize signal paths and trace errors using the integrated or standalone Path Viewer.
- Trace clock and reset origins/paths using the integrated Clock/Reset Tree Browser.
- Obtain help about violated rules in HTML format, including, in some cases, circuit diagrams and HDL code examples that explain the problems.
- Hyperlink directly from the Error Viewer to the suspect code in your HDL files, correct the errors (highlighted in the source files), and rebuild your design.
- Run design queries on your elaborated design database from the integrated Tcl shell, using a predefined set of Tcl procedures (API).

Leda provides an integrated debugging environment. Changes you make in one mode (GUI, batch, Tcl shell) are automatically reflected in all modes for that session. And the different views available in the GUI (Path Viewer, Clock/Reset Tree Browser, source code hierarchy, modules/units, Error Viewer) are synchronized to make it easier for you to figure out what's wrong and fix it.

Leda Terminology

There are several terms that have specific meaning in the context of using Leda. It is a good idea to familiarize yourself with this terminology before using Leda. [Table 2](#) lists the key Leda terms and their definitions.

Table 2: Key Terminology in Leda

Term	Definition
Rule	An expression written in VRSL (for VHDL) or VerSL (for Verilog) that precisely describes and constrains an HDL construct. Rules can either be in source code form, which is ASCII text, or compiled, after you build them using the Specifier. There are two basic kinds of rules: coding rules and design rules. You write coding rules in VerSL/VRSL and design rules in Tcl or C.
Ruleset	A collection of one or more rules. You write rule source code in rulesets, which are stored in <i>ruleset.rl</i> (for VRSL) or <i>ruleset.sl</i> (for VerSL) files. You write rulesets as plain text files that follow prescribed VRSL or VerSL coding conventions. Leda prepackaged rules are organized into the major subdivisions of various coding standards. For example, in the RMM policy based on the <i>Reuse Methodology Manual</i> , there are rulesets for coding for portability and guidelines for clocks and resets. Note: Rulesets are used both for coding rules that you develop in VRSL or VerSL and design rules that you develop in Tcl or C using the supplied APIs.
Policy	A policy can contain any number of rulesets. Leda uses policies to organize major coding standards such as the RMM coding guidelines or the IEEE synthesis subsets. To see how policies are used to organize the Leda prepackaged rules, cd to the \$LEDA_PATH/rules directory and review the policy source files (for example, rmm or ieee_synthesis) or review the <i>Leda Prepackaged Rules Guides</i> , which provide detailed reference information for all of the Leda prepackaged rules. There is a separate PDF file for each policy in the \$LEDA_PATH/doc directory.
Template	A predefined model of an HDL coding construct. Leda scans your HDL code to find segments that match templates used in the rules that you select. Each template in the VRSL and VerSL rule specification languages is either primary or secondary. They are all clearly labelled in the VRSL Reference Guide and VerSL Reference Guide , which provide complete reference information for all templates and attributes.
Attribute	Aspects or characteristics of templates that you use to define exactly the kind of HDL code a template will match. Each template has a defined set of attributes that you can use with it. You can use some attributes as templates, with their own sets of attributes. VRSL and VerSL are flexible in this way. Templates and attributes are the building blocks that you use to write rules.

Table 2: Key Terminology in Leda (Continued)

Term	Definition
VRSL	VHDL rule specification language. VRSL is a macro-based language which you use to write rules that check VHDL source code for deviations from prescribed standards. Note that VRSL is not an HDL.
VeRSL	Verilog rule specification language. VeRSL is a macro-based language which you use to write rules that check Verilog source code for deviations from prescribed standards. Note that VeRSL is not an HDL.

Types of Leda Rules

There are four general types of Leda rules, as shown in [Table 3](#).

Table 3: Types of Leda Rules

Rule Type	Description
Block-level rules (aka Coding rules)	Block-level rules constrain different HDL constructs by ensuring that they correspond to acceptable values, ranges, or templates. This means that you can define a syntactic/semantic subset of the language that is uniquely targeted to your design flow and methodology. For example, you can use Leda to check for HDL constructs such as architecture, body, module instantiation, and variable assignments.
Chip-level rules (aka Hardware rules)	<p>Chip-level rules control the hardware semantics of VHDL and Verilog. Certain HDL constructs result in specific hardware features when you synthesize the descriptions. For example:</p> <ul style="list-style-type: none"> VHDL—<code>ck='1'</code> and <code>ck'event</code> represent clocks active on the rising edge Verilog—<code>@posedge clk</code> represents a clock active on the rising edge <p>You can check for the proper use of clocks, and inferred hardware such as latches, flip-flops, and finite state machines. You can also check for tristated signals, asynchronous feedback loops, and other HDL design concerns.</p> <p>Chip-level checkers are not programmable, but configurable to some extent. It includes a set of rules that cannot be modified by the user. It is fast, efficient but not as open as the netlist checker.</p>

Table 3: Types of Leda Rules

Rule Type	Description
Design rules (aka Netlist rules)	<p>Design rules work on the entire design hierarchy. For design rules to work, you must specify a -top module in the design hierarchy.</p> <p>Netlist checkers are less efficient than the chip-level checker, but it is completely open (i.e. customers can program their own check in Tcl or in C). It includes much more checks than chip-level checkers.</p> <p>In some cases, netlist rules provide a better flexibility than old chip-level rules, for instance when checking for clock domain crossing (NTL_CLK05/C_1202), only NTL_CLK05 can be configured not to fire on synchronous clocks, with C_1202, there is no way to inform Leda that some clocks are synchronous hence the rule shall not fire.</p> <p>For more information, see the “Checkers Tab” on page 111</p>
SDC rules	Synopsys Design Constraint (SDC) rules check SDC files for internal consistency and consistency with the design.

Approaches to Using Leda

There are several ways to use Leda, depending on your needs. [Figure 2](#) provides an overview of the recommended approaches for using Leda’s coding rules. You can also develop design rules that run against the elaborated design database using the supplied Tcl and C APIs. For more information, see the [Leda Tcl Interface Guide](#) or [Leda C Interface Guide](#). In the following diagram, the referenced sections in this manual explain how to proceed.

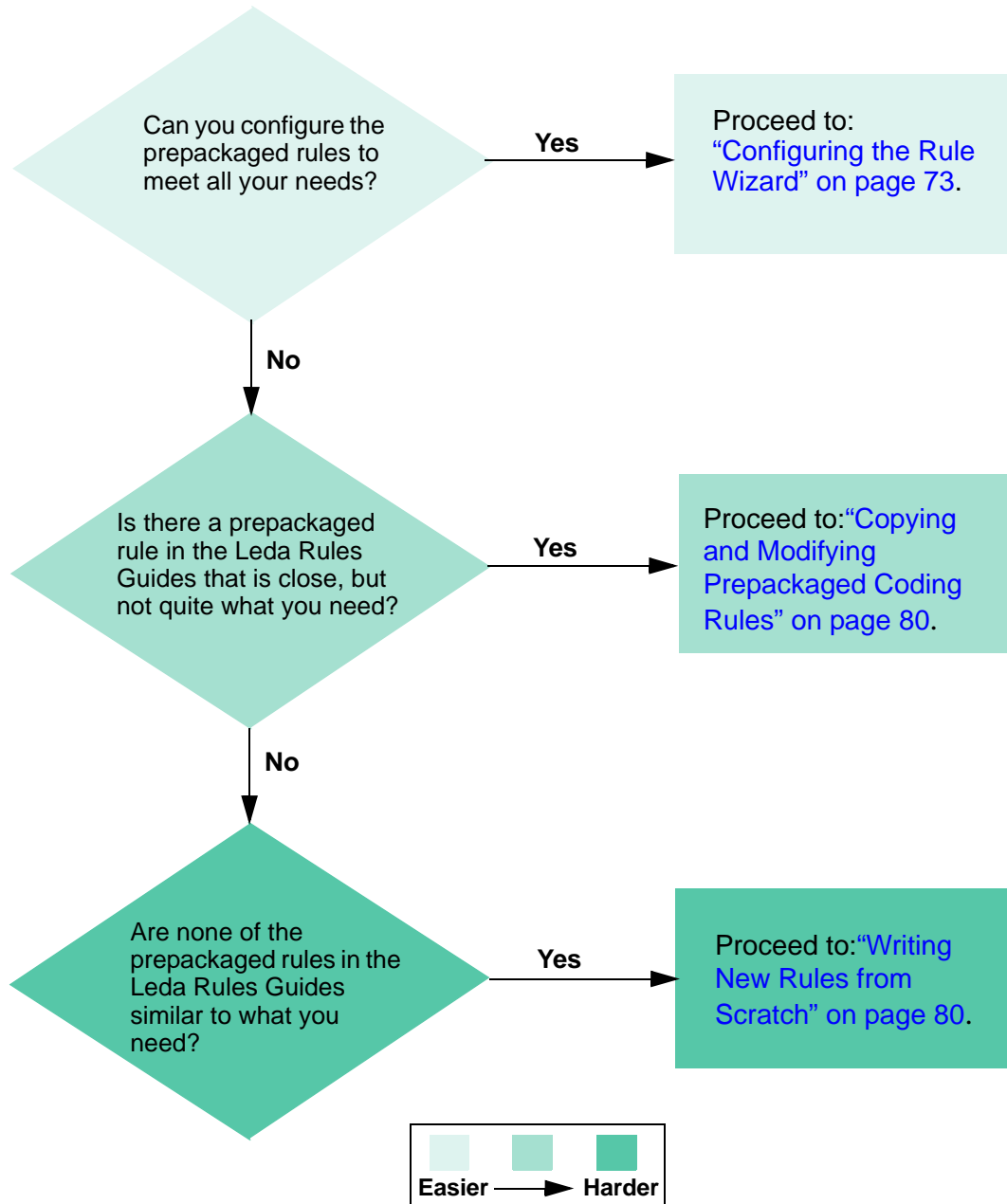


Figure 2: Approaches to Using Leda

Using Leda in Batch, GUI, and Tcl Shell Modes

You can use Leda in batch, GUI, and Tcl shell modes, and switch back and forth between any of these modes in the same session depending on the task at hand. Changes you make in one mode with Leda are automatically reflected in all other modes. Leda's different modes all work together to help you debug and fix HDL coding and chip-level design issues (see [Figure 3](#)).

Batch mode is best for experienced users because you can run scripts that reuse established command-line options. GUI mode is a good option for new users because of the more intuitive interface. After you run a check on your design files in either mode, you can use the GUI to review and debug errors. When you are in GUI mode, you also have access to the Tcl shell using the console at the bottom of the main window. You can use the Tcl shell to interactively check design rules against your elaborated design database using a Design Query Language (DQL). Leda provides an API with a set of Tcl procedures that you can use to call the DQL (for example, `get_all_clock_origins`). For more information, see the [Leda Tcl Interface Guide](#). You can also use the Tcl shell to manage your Leda projects, rule configurations, and runs with the Checker (see “[Using Leda Tcl Shell Mode](#)” on page 187).

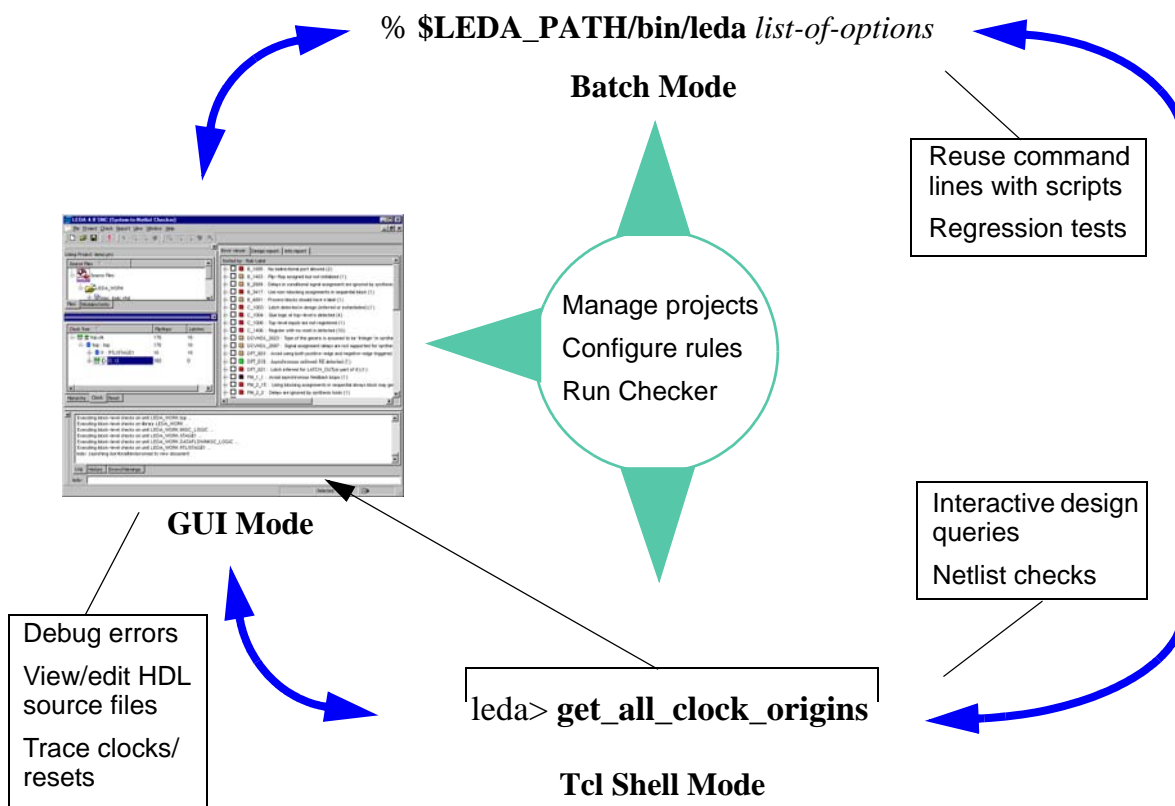


Figure 3: Leda Modes of Operation

Invoking Leda

To invoke Leda in batch mode:

```
% leda batch_command_line_args/options [-project project_name]
```

To invoke Leda in GUI mode, type `leda` with no arguments:

```
% leda
```

To invoke Leda in Tcl shell mode:

```
% leda +tcl_shell [-project project_name]
```

```
% leda +tcl_shell batch_command_line_args/options [-project project_name]
```

If you don't specify a `project_name` or any batch command-line arguments or options, Leda does not create or open a project for you. However, in GUI or Tcl shell mode, when you don't specify a `project_name`, but include batch command-line arguments or options, Leda assumes an implicit project named "leda". Thus, the following two commands are equivalent:

```
% leda +tcl_shell batch_command_line_args/options
```

```
% leda +tcl_shell batch_command_line_args/options -project leda
```

GUI mode includes Tcl shell mode using the `leda>` prompt at the bottom of the main window. There is no difference between commands executed in Tcl shell mode and GUI mode, except that GUI-specific commands (that is, all commands starting with the prefix "gui_" except `gui_start`) are invalid outside GUI mode.

Switching Modes

To go from Tcl shell mode to GUI mode, at the Tcl shell prompt, type:

```
leda> gui_start
```

To go from GUI mode to Tcl shell mode, in the Tcl shell console at the bottom of the main window, type:

```
leda> gui_stop
```

A GUI command line is always equivalent to a Tcl shell command line with the same arguments and options, immediately followed by the execution of the `gui_start` command at the Tcl shell prompt. For example:

```
% leda +gui [-project project_name]
```

is equivalent to

```
% leda +tcl_shell [-project project_name]
```

```
leda> gui_start
```

and

```
% leda +gui batch_command_line_args/options [-project project_name]
```

is equivalent to

```
% leda +tcl_shell batch_command_line_args/options [-project project_name]
```

```
leda> gui_start
```

Creating Projects

You can create a project in any mode and reuse that project in any other mode. To create a new project:

- In GUI mode, choose **Project > New** from the pulldown menu. This launches the Project Creation Wizard. Follow the steps indicated. For more information, see [“Creating Projects to Check HDL Code” on page 91](#).
- In Tcl shell mode, execute the `project_new` command followed by other project specification commands (with prefix “`project_`”).

```
leda> project_new project_name
```

- In batch mode, use the `-project` option and specify a new project name:

```
% leda batch_command_line_args/options -project new_project_name
```

Opening Projects

To open a project:

- In GUI mode, choose **Project > Open**.
- In Tcl Shell mode, execute the `project_open` command followed by the project name:

```
leda> project_open project_name
```

- In batch mode, use the `-project` option and specify an existing project name

```
% leda [+gui | +tcl_shell] -project project_name
```

Enabling Design Query Commands

The Tcl interpreter, embedded in the GUI and the Tcl Shell, supports a set of commands that you can use to query the elaborated design database. For this to work, you first have to elaborate the design. You can do this several ways:

- In batch mode, specify a `-top` unit:

```
% leda hdl_file_list -top my_top_unit +tcl_shell
```


- In GUI mode, choose **File > Preferences** and select the “Netlist Checks” check box in the Checker category. Then execute the Checker with some netlist checking rules selected (for example, prepackaged rules from the NETLIST policy).
- In Tcl mode, use the elaborate command after you invoke the Tcl shell and open an existing project:

```
% leda +tcl_shell
leda> project_open existing_project_name
leda> elaborate
```

- In Tcl mode, you can also read in some HDL source files, specify the top-level unit, and link/elaborate the design as follows without specifying a project:

```
% leda +tcl_shell (to start the tool)
leda> read_verilog netlist.v (or a set of files)
leda> current_design name_of_top_level_unit
leda> elaborate (resolve all instantiations and elaborate the design)
```

With your elaborated design in memory, you now can:

- Execute interactive queries using the Design Query Language (DQL) functions. For complete reference information on the Tcl DQL API, see the [Leda Tcl Interface Guide](#).
- Execute built-in Tcl commands (see “[Built-in Tcl Commands](#)” on page 189), or load a Tcl script that contains a series of built-in Tcl commands.

Configuring the GUI

There are additional Tcl commands that you can use to manage/configure the GUI. They all start with the “gui_” prefix. For details, see the help in the Tcl shell:

```
leda> help
```

From the list of commands returned, pick the one you are interested in and then get the help on that command. For example:

```
leda> gui_toggle_summary -help
```

All other commands supported by the Tcl interpreter are accessible any time the Tcl shell is running.

Typical Leda Usage Scenarios

You may want to switch back and forth between batch, GUI, and Tcl shell mode depending on the task at hand. To better illustrate some typical usage scenarios, following are some case studies that show effective ways to use Leda in all three modes. The HDL files in these case studies are referenced as \$DES2001. Let's say that this directory contains three subdirectories: two including Verilog files (DW_usbd.v and misc.v) and one including VHDL files (dw_8051.vhd).

Case Study 1

This case study shows how to check a design in batch mode, debug errors in GUI mode, and execute design queries from the Tcl shell within the GUI:

```
% leda $DES2001/*/*.v -top DW_usbd_chip -project dw_usb
% leda -project dw_usb
```

or equivalent

```
% leda +gui -project dw_usb
```

The first line in this example executes the Checker in batch mode and creates a project that includes the HDL files that you want to check.

The second line invokes the GUI with the project created in the first line. From the Tcl prompt at the bottom of the GUI window, execute the elaborate command to enable the DQL, and then use the get_all_clock_origins query as follows:

```
leda> elaborate
leda> get_all_clock_origins
```

In this particular design, the get_all_clock_origins command returns two primary clocks (clkref and clk_48). After this command runs, you can see the clock signals displayed in the clock tree browser on the left side of the GUI main window.

Note that in this example, you must first execute the elaborate command before browsing the design, because it is not loaded in memory when the GUI opens. This is because the design elaboration/checking and the GUI opening were executed by two separate command lines or processes. [Case Study 2](#) uses a single command line (one process), so it does not require an elaboration in a second step.

Case Study 2

Check a design in batch mode, debug errors in GUI mode, and execute design queries from the Tcl Shell within the GUI, but with one initial command line:

```
% leda $DES2001/*/*.v -top DW_usbd_chip -project dw_usb +gui
```

In this example, the Checker executes in batch mode and creates the project before automatically opening the GUI. From the Tcl prompt at the bottom of the GUI window, you can now execute the `get_all_clock_origins` query as follows:

```
leda> get_all_clock_origins
```

As before, with this particular design, the `get_all_clock_origins` command returns two primary clocks (`clkref` and `clk_48`).

Case Study 3

Check a design in batch mode, and execute design queries in Tcl shell mode without using the GUI:

```
% leda $DES2001/*/*.v -top DW_usbd_chip -project dw_usb
% leda -project dw_usb +tcl_shell
```

The first line in this example executes the Checker in batch mode and creates a project for the HDL files.

The second line invokes Tcl shell mode with the project created in the first line. At the Tcl prompt, execute the `elaborate` command and then the `get_all_clock_origins` query as follows:

```
leda> elaborate
leda> get_all_clock_origins
```

As before, with this particular design, the `get_all_clock_origins` command returns two primary clocks (`clkref` and `clk_48`).

Note that, as in [Case Study 1](#), you must first execute the `elaborate` command before browsing the design, because it is not loaded in memory when the Tcl shell opens. [Case Study 4](#) uses a single command line (one process), so it does not require an elaboration in a second step.

Case Study 4

Check a design in batch mode and invoke the Tcl shell in one command line. Execute design queries from the Tcl shell. Then invoke the GUI, observe the project, execute the same design queries, and go back to Tcl shell mode.

```
% leda $DES2001/*/*.v -top DW_usbd_chip -project dw_usb +tcl_shell
```

This line executes the Checker in batch mode, creates a project on the fly, and opens Tcl shell mode. Now, at the Tcl prompt in the Tcl shell mode, execute the DQL command `get_all_clock_origins`:

```
leda> get_all_clock_origins
```

As before, with this particular design, the `get_all_clock_origins` command returns two primary clocks (`clkref` and `clk_48`).

Now, invoke the GUI using the `gui_start` command:

```
leda> gui_start
```

The GUI automatically opens the project “`dw_usb`”. Check the project name in the upper-left of the main window just below the toolbar. Notice that the clock tree browser reports the two same primary clocks. Then use the Tcl prompt at the bottom of the main window to execute the same DQL command: `get_all_clock_origins`. As before, with this particular design, the `get_all_clock_origins` command returns two primary clocks (`clkref` and `clk_48`).

Then exit the GUI using the `gui_stop` command:

```
leda> gui_stop
```

You can type this command either at the Tcl prompt in the GUI or at the `leda>` prompt in the shell where you invoked Leda.

When the GUI exits, you are back in Tcl shell mode. Execute the same `get_all_clock_origins` DQL command again. The command still returns the same two primary clocks (`clkref` and `clk_48`) for this design, proving that the design is still in memory.

Case Study 5

This example is the same as [Case Study 4](#), except that no project name is given:

```
% leda $DES2001/*/*.v -top DW_usbd_chip +tcl_shell
```

After invoking Leda this way, if you follow the same steps as in [Case Study 4](#) you get the same results, except that the project name opened by the GUI is “`leda`”. Leda creates an implicit “`leda`” project automatically when you don’t specify a project using the `-project` option.

Case Study 6

Check a design in batch mode, debug the errors in GUI mode, and execute design queries from the Tcl Shell within the GUI. Then exit the GUI to Tcl shell mode and execute the same design queries again from the Tcl shell.

```
% leda $DES2001/*/*.v -top DW_usbd_chip -project dw_usb
```

```
% leda +gui -project dw_usb
```

In this example, the first line executes the Checker in batch mode and creates a project named `dw_usb`. The second line opens the GUI with the project created in the first line. From the Tcl prompt at the bottom of the main window, execute the `elaborate` command to enable the DQL database and then run the `get_all_clock_origins` query:

```
leda> elaborate
leda> get_all_clock_origins
```

As before, with this particular design, the `get_all_clock_origins` command returns two primary clocks (`clkref` and `clk_48`). As in [Case Study 1](#), you must first execute the `elaborate` command before browsing the design because it is not loaded in memory when the GUI opens.

Now exit the GUI and switch to Tcl shell mode using the `gui_stop` command:

```
leda> gui_stop
```

The GUI quits, and you are returned to Tcl shell mode in the shell where you invoked Leda. At the `leda>` prompt, execute the `get_all_clock_origins` DQL command again. The command returns the same two primary clocks (`clkref` and `clk_48`) for this particular design, proving that the design, which was elaborated from within the GUI, is still in memory when you switch to Tcl Shell mode.

Case Study 7

Invoke Leda in Tcl shell mode, open a project, and then launch the GUI.

```
% leda +tcl_shell
```

This enters Tcl shell mode. Check that no project or no design is loaded by trying to execute the `get_all_clock_origins` DQL command:

```
leda> get_all_clock_origins
```

This time, the query returns an error message saying that this “command is unknown”. Now open the project created in the [Case Study 6](#), elaborate the design, and try to get the clock origins:

```
leda> project_open dw_usb
leda> elaborate
leda> get_all_clock_origins
```

Now that you have elaborated the design, the `get_all_clock_origins` command returns the two primary clocks (`clkref` and `clk_48`) for this particular design.

Open the GUI using the `gui_start` command:

```
leda> gui_start
```

Again, the GUI automatically opens the `dw_usb` project. From the Tcl prompt at the bottom of the main window, execute the `get_all_clock_origins` DQL command again. The command returns the two primary clocks (`clkref` and `clk_48`). Check that the clock tree browser displays the same information.

Case Study 8

Open Leda in GUI mode and open a project at the Tcl prompt from within the GUI.

```
% leda
```

or equivalent

```
% leda +gui
```

Both of these lines invoke Leda in GUI mode. Check that no project or no design is loaded by trying to execute the `get_all_clock_origins` DQL command from the Tcl prompt at the bottom of the main window:

```
leda> get_all_clock_origins
```

This command returns an error message saying that this “command is unknown”. Now open the project created in [Case Study 6](#), again from the Tcl prompt at the bottom of the main window:

```
leda> project_open dw_usb
```

The GUI opens the project. Now, elaborate the design and try the `get_all_clock_origins` query again:

```
leda> elaborate
```

```
leda> get_all_clock_origins
```

The command returns the two primary clocks (`clkref` and `clk_48`) for this particular design.

Case Study 9

Check a design using Tcl commands and no project:

```
% leda +tcl_shell
```

This enters Tcl shell mode. Check that no project or no design is loaded by trying to execute the DQL command `get_all_clock_origins`:

```
leda> get_all_clock_origins
```

This should return an error message saying that this command is unknown. Now read files to check and elaborate the design with the following commands:

```
leda> read_verilog $DAC2001/*/*.v  
leda> elaborate -top DW_usbd_chip
```

Now verify that the design is correctly loaded by executing the DQL command:

```
leda> get_all_clock_origins
```

This command returns the two primary clocks (clkref and clk_48) for this particular design.

About Design Rules

Design rules are applied to the entire design hierarchy, whereas block-level rules are applied to each unit individually. You can write design rules using the templates listed below. Other templates, such as clock and synchronous_reset, also contain attributes that you can use to write chip-level rules:

- Design Template
- Connectivity Template
- Test Signal Template
- Data Signal Template
- Flipflop Template
- Latch Template

For detailed reference information on all VeRSL and VRSL templates and attributes, see the [VeRSL Reference Guide](#) (for Verilog) and [VRSL Reference Guide](#) (for VHDL).



Note

You can also write design netlist-checking rules using Leda's Tcl or C rule APIs. For more information, see the [Leda Tcl Interface Guide](#) or [Leda C Interface Guide](#).

Using .db Files for Checks

A .db file is a technology-dependent representation of a library used by several Synopsys tools. In particular, .db files are used by Design Compiler and Path Mill. A .db file is generated by Library Compiler from a .lib file that contains the source representation of the information. Many gate-level or netlist representations include .db files. Leda reads .db files in order to correctly check chip-level rules in designs that include .db files. Leda reports any errors found at the file instantiation level.

Leda uses .db files similarly to the way Design Compiler uses them. (Note that Path Mill uses a different environment variable.)

If you are using a .db file and want Leda to read this file for checking chip-level rules, set the `link_library` environment variable to the location of your .db file, as shown in the following example:

```
% setenv link_library gtech.db
```

Set the `search_path` environment variable to the location of your db libraries, as shown in the following example:

```
% setenv search_path /synopsys/2001.08-Synthesis/libraries/syn
```

You can specify multiple `link_libraries` and `search_paths` by separating the names with spaces and enclosing the list of entries in quotation marks. For example:

```
% setenv link_library "gtech.db class.db"
```



Note

The `search_path` variable is only useful for pointing to link libraries, not design files.

The `link_library` and `search_path` variables are the same variables used with Design Compiler. When these variables are set, Leda tries to resolve the instances with an algorithm that depends on whether the source code is in VHDL or Verilog. If Leda cannot find an architecture/module, it searches the .db files specified with the `$link_library` variable. If found, Leda generates hardware information for the instantiated unit. If not found, Leda treats the instantiated unit as a black box. For more information about using Leda with .db files, including the limitations, see [“Using .db Files for Checks” on page 39](#).

Limitation with Gates in .db Files

For instantiated gates in .db files, Leda does not handle composite ports in .db cells or designs. Leda recognizes the instantiation if the port name matches the specified formal name, but does not recognize it as a bus. Leda does not perform type verification in such cases. For example, Leda accepts the following VHDL architecture definition:

```
architecture a of top is
  component GATE is
    port ( A : in bit;
          B : in bit;
          Z : out bit );
  end component;
  signal input1, input2 : bit;
  signal ouput : bit;
begin
  INST: GATE port map ( A => input1, B => input2, Z => output);
end;
```

where the cell GATE has the following ports, Leda considers Port B to be a port of type bit.

```
A
B[0]
B[1]
B[2]
B[3]
Z
```

About Hardware-Based Rules

Verilog and VHDL are often used in environments that add hardware-specific semantics to the code. For example, wait statements, clocks, and asynchronous/synchronous expressions must be carefully defined for synthesis tools. You can use Leda to write rules using hardware-specific attributes to govern the most common hardware-specific semantics of Verilog and VHDL just as easily as coding rules. The following sections explain some of the hardware inference semantics that Leda uses to implement hardware-based rules:

- [“Finite State Machine Rules” on page 42](#)
- [“Set and Reset Detection in VHDL and Verilog” on page 49](#)



Attention

Leda’s hardware-based rules are designed to work on synthesizable HDL code. If you run the Checker on a project that includes non-synthesizable HDL code (for example, a testbench), the results are unpredictable. To solve this problem, you can mask your testbench code using the Synopsys `synthesis_off` and `synthesis_on` or `translate_on` and `translate_off` directives or pragmas. For information on setting these directives, see [“Creating Projects to Check HDL Code” on page 91](#) if you are using the GUI Checker or [Table 17 on page 148](#) if you are using the command-line Checker.

Finite State Machine Rules

Leda can infer finite state machines (FSMs) in Verilog or VHDL designs and apply rules that you select to define the kind of FSM that is acceptable (for example, Moore or Mealy). Leda can only identify FSMs that are coded using case statements to define the states and transitions. Leda cannot identify FSMs coded using “if” statements.

Leda recognizes 1-process, 2-process, and 3-process FSM models if all the processes are in the same block (architecture or module).

Leda comes with a set of prepackaged FSM rules in the State Machines ruleset, which is part of the Leda General Coding Guidelines policy. For details, see the [Leda General Coding Rules Guide](#). There are also special “fsm” templates in VRSL and VerSL that you can use to develop custom rules for FSMs.

Hardware Inference

This section explains how Leda identifies entities like primary clock, generated clock, gated clock, clock origin, and clock domain.

Primary Clock

A primary clock is a primary input port used as a clock. Leda also considers inverters and buffers on the clock path. Some examples of primary clocks are shown in Figure 4, 5 and 6.

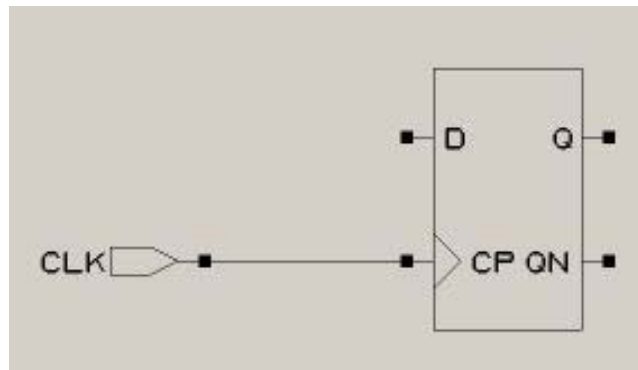


Figure 4: Signal CLK is a primary clock

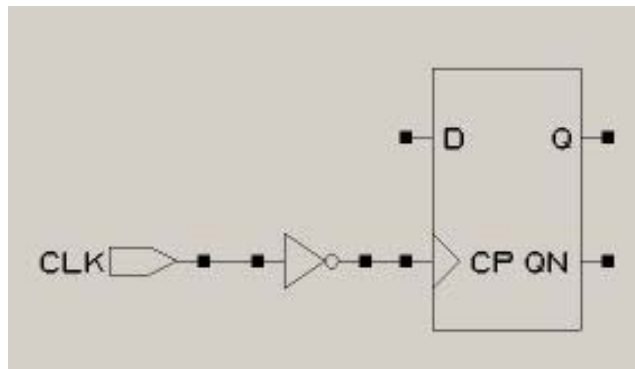


Figure 5: Signal CLK is a primary clock

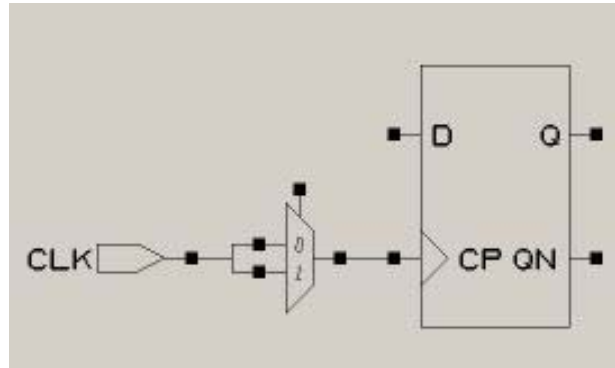


Figure 6: Signal CLK is also a primary clock

Internally Generated Clock

An internally generated clock is a clock derived from a primary clock or an internal signal driving the clock input of a flip-flop but not connected to a primary port.

In Figure 7, the internally generated clock has a synchronous relationship with its primary (or master) clock.

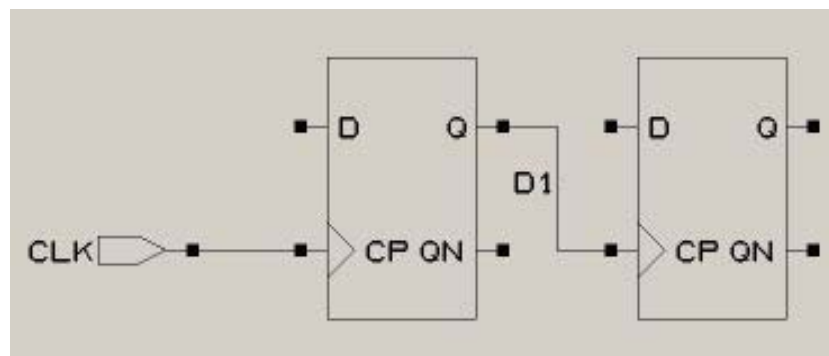


Figure 7: Signal D1 is a generated clock from primary clock CLK

In Figure 8, signal INT1 is a generated clock as there is no connection to any primary port.

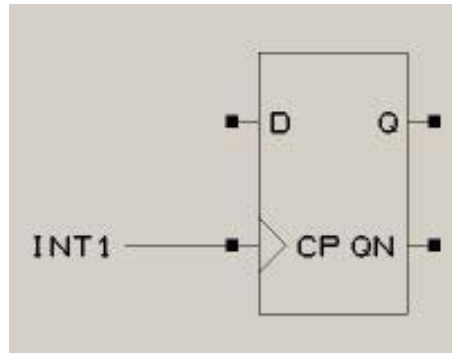


Figure 8: INT1 is a generated clock as no connection to a primary port

Figure 9 models a disconnected signal driving the clock input of a flip-flop. This implies that for any disconnection in the clock connectivity, Leda infers a new clock.

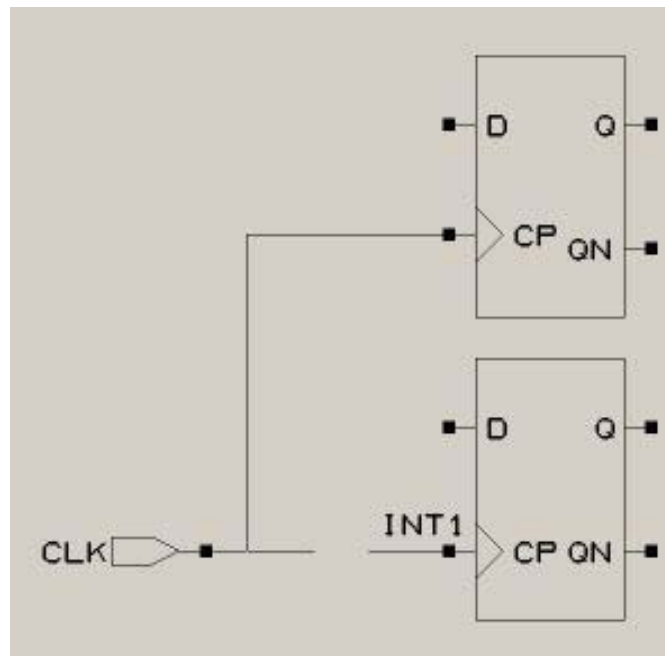


Figure 9: INT1 is a generated clock due to disconnection

Gated Clock

A gated clock is a signal on a clock path that is the output of a combinatorial complex block. A gated clock is modelled in both Figure 10 and 11. In Figure 10, the clock origin is CLK and in Figure 11, the gated clock (output signal of the AND gate) is the clock origin.

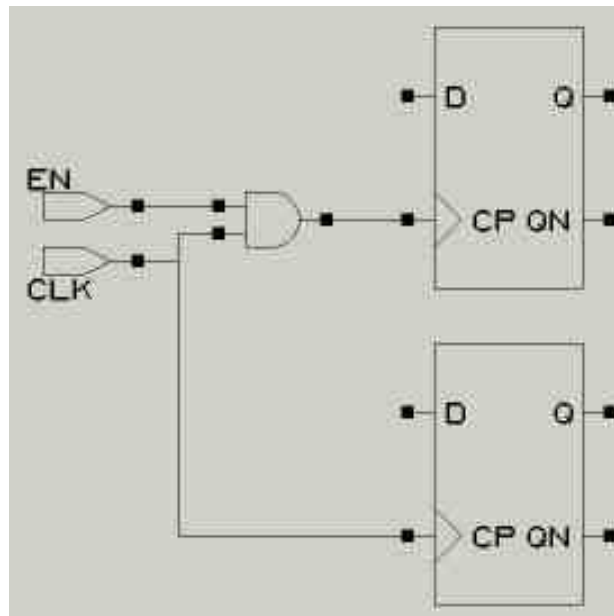


Figure 10: Gated clock

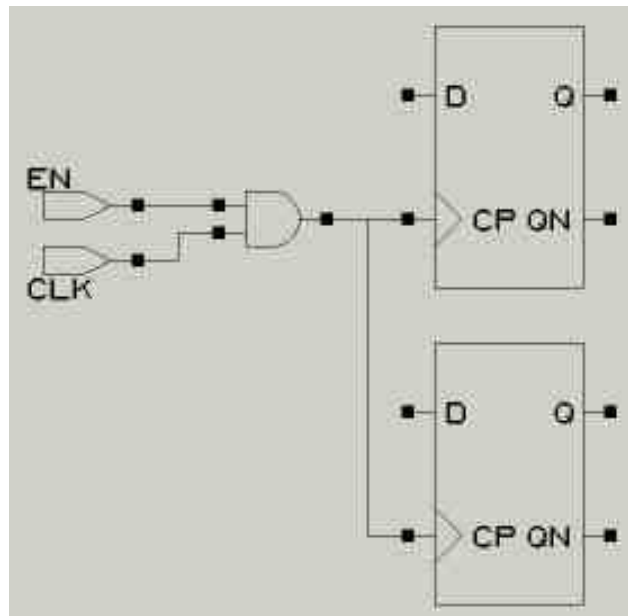


Figure 11: Gated clock



Attention

Figure 11 infers an internally generated clock as there are no identified clock origins (primary clock or internally generated clock) on the input cone to the gate.

The general rule for inferring gated clock is as follows:



Note

A gated clock is considered as a clock origin, if and only if, there is zero or more than one identified clock origins in its fan-in cone. Such gated clocks, that is also a clock origin are considered as a kind of internally generated clock. Other gated clocks having exactly one clock origin in its fan-in cone, are neither considered as clock origins nor as internally generated clocks.

Clock Origin

Clock origins are the set of all clocks comprising the primary clocks and internally generated clocks.

**Note**

If a clock or reset/set origin signal is tied to a constant value, then that signal no longer act as a clock or reset/set regarding its register. So, the signal is not stored in the control origin list.

Clock Domain

A clock domain is a set of flip-flops driven by the same clock origin or derivatives of this clock origin. A derivative is an internally generated clock whose master is the current clock origin or is a gated clock and one of the inputs to the gate is the current clock origin. The master clock origin of a clock domain is the highest clock in the hierarchy. All clocks in a clock domain are synchronous.

Set and Reset Detection in VHDL and Verilog

Leda supports complex set and reset detection in both VHDL and Verilog designs. In releases prior to 4.0, Verilog resets required the assigned value to be constant (or at least globally static), whereas VHDL resets accepted dynamic values. Thus, in VHDL, loads were treated as resets. Now, for VHDL and Verilog, a set or reset is required to assign a globally static or constant value. This means that in VHDL some signals that Leda previously identified as resets are no longer considered to be resets when the value is not constant.

Leda detects both block-level constants and complex sets and resets. For example, Leda detects set and reset signals in code like the following:

```
if RST then
    S <= '0';
elsif CK'event and CK='1' then
    S <= D
end if;
```

Leda also detects resets in code with a complex flow of control. In this case, reset detection is enabled if Leda finds a reset embedded in a hierarchy of conditional if or case statements. Leda also detects implicit resets in code like the following:

```
always @(posedge rd_clk)
begin : READ_MEM_PROC
    dp_data_rda <= 0;
    if (~dp_csa_n) begin
        dp_data_rda <= memory[dp_addr];
    end
end
end
```

In the above example, `dp_data_rda` has a reset expression, which is `dp_csa_n` (`~(~dp_csa_n)`).

Block-Level Constant Detection in Verilog and VHDL

Leda determines that an object (signal, reg, or net) is tied to a static value (constant) if either of the following are true:

- It is a supply
- It is assigned a constant value, no other assignment is performed on the object, and the single assignment is not controlled by an expression (for example, if or case statement).

Leda determines that an expression is static (constant) if either of the following are true:

- It is the primary expression denoting an object tied to a static value
- It is an expression that includes a globally static expression

Leda takes the detection of an object tied to a constant value into account in its reset detection logic. For example, in the following example, the signal Reset is detected as a reset:

```
Zero<='0';
  process (CK, Reset)
  begin
    if (Reset='0') then S<=Zero;
    else if Ck'event and Ck='1' then S<=D end if;
  end process;
```

**Note**

Leda does not detect static values in complex expressions.

Rules Leda Cannot Check

There are certain kinds of rules that Leda cannot specify or check. Before you get started customizing prepackaged rules or creating new ones, consider the following limitations. Leda does not support rules that:

- Check for things that you cannot define. For example, Leda cannot check to make sure HDL comments are written in English or any other language.
- Check language-based, chip-level design issues. For example, Leda cannot check that the same constant is used in different modules, and therefore should really be declared externally.

Finally, Leda coding rules must use the templates and attributes that are part of the VRSL and VerSL rule specification languages (see [Approaches to Using Leda](#)). This means that you cannot create a new template or attribute on your own. There is no such restriction on design or SDC rules that you create using the supplied Tcl and C APIs. For more information, see the [Leda Tcl Interface Guide](#) or [Leda C Interface Guide](#).

2

Writing and Checking HDL Designs

Introduction

This chapter explains how to write and check designs that contain VHDL, Verilog, or a combination of VHDL and Verilog code. The way the Leda Checker operates on these three types of designs depends on how closely you follow the VHDL and Verilog Language Reference Manuals (LRMs) in your code. Depending on how you set up the Checker, you can get compiler warnings for code that does not strictly adhere to the LRMs. In some cases, this is not desirable if you use a non-standard coding style that is needed for important downstream applications. Leda allows you to make “semantic exceptions” for cases like this using a simple button in the Checker tool.

This chapter explains a few of the basics for designing in VHDL and Verilog, and provides examples of “semantic exceptions” that you should be aware of so that you can enable or disable this setting prior to running the Checker. In addition, data and type mappings for mixed-language designs are explained in detail. The last section explains Leda’s support levels for Verilog 2001 constructs. This information is presented in the following major sections:

- [“Writing & Checking VHDL Designs” on page 52](#)
- [“Writing & Checking Verilog Designs” on page 56](#)
- [“Writing & Checking Mixed-Language Designs” on page 59](#)
- [“Mapping Data Types” on page 60](#)
- [“Verilog 2001 Support” on page 65](#)
- [“SystemVerilog Support” on page 65](#)
- [“Clock Grouping Feature” on page 66](#)
- [“Netlist Reader” on page 68](#)

Writing & Checking VHDL Designs

You write VHDL code in design entities. There are five types of design entities: package, entity, configuration, package body, and architecture. VHDL code must respect one of two standards: VHDL 87 or VHDL 93. All design entities in one design must use the same mode. These modes are formally defined in the two VHDL Language Reference Manuals (LRMs). To ensure compatibility with an LRM, you must first compile or analyze the code. If compilation is successful, Leda stores the unit in a library. You must provide the library's name. The physical representation of a library is usually (but not necessarily) a directory. A VHDL design can contain many libraries organized hierarchically. In Leda's terminology, this is called a VHDL project. Libraries are divided into working libraries and resource libraries.

The difference between a working library and a resource library is that you use the former to store the design units being developed, whereas the latter contains shared resources such as standard packages or leaf cells. Examples of resource libraries are `STD`, `IEEE`, `GTECH`, `SYNOPSIS`, `VITAL` and so on. You do not modify these libraries, but you may need to reference them from your working libraries. The Checker only checks the code in those libraries marked as working libraries.

If you want to apply chip-level checks to your design, you must first elaborate the project. This means that you must resolve the connectivity between all design units. If you instantiate one unit in another unit, the elaborator looks for the design unit that represents the instantiated unit, and connects the ports as indicated by the instantiation. For more information on VHDL elaboration, see the VHDL LRM. The Checker also applies a hardware inference algorithm to build an image used to validate hardware rules.

VHDL Semantic Exceptions

There are a number of semantic exceptions in VHDL. By default, the Checker observes these semantic exceptions. You can deselect this setting using the Project Update Wizard that comes up when you choose **Project > Edit** from the Checker's main menu. Click on the VHDL tab in this window, and deselect the "With semantic exceptions" check box at the top left. Following are several VHDL code examples that cause compiler errors unless you have "With semantic exceptions" enabled prior to running the Checker.

VHDL Semantic Exception—Example 1

When you compile a file with the semantic exceptions enabled, you use special visibility rules for operators that do not conform to the standard rules defined in the VHDL LRM, but are used by different commercial applications. Consider the following VHDL code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
package COMP_LOGIC_1164 is
    function "=" (A,B: STD_ULOGIC) return BOOLEAN ;
    function "/=" (A,B: STD_ULOGIC) return BOOLEAN ;
end package COMP_LOGIC_1164;

library IEEE;
use IEEE.STD_LOGIC_1164.all; -- imports implicit comparison operators
use WORK.COMP_LOGIC_1164.all; -- imports explicit comparison operators
architecture ARC of ENT is
    signal A,B,C : STD_LOGIC;
begin
    C <= '1' when A = B else '0'; -- illegal by the LRM
end;
```

If you compile the architecture ARC above with the standard visibility, the compiler generates an error message for the signal assignment. The problem is that the operator call “=” is ambiguous between the operator “=” implicitly defined with the type `STD_ULOGIC` in the `STD_LOGIC_1164` package and the operator “=” explicitly defined in the `COMP_LOGIC_1164` package.

If you compile the architecture ARC with semantic exceptions enabled, explicit operators are given a higher priority in the visibility rules than implicit operators, which allows this VHDL code to compile successfully using the explicit operator in the `COMP_LOGIC_1164` package.

VHDL Semantic Exception—Example 2

Do not hide library names using other declarative items such as a component names. For example, the following code causes a compilation error unless you have semantic exceptions enabled prior to checking your design:

```
entity E is
end;
library FOO; -- = WORK
entity TOP is end;
architecture RTL of TOP is
  component FOO
  end component;
  for A : FOO use entity FOO.E;
-- in special mode, FOO after entity keyword is the library
-- and not the component
begin
  A : FOO;
end;
```

VHDL Semantic Exception—Example 3

Do not hide library names using unit names. For example, the following code causes a compilation error unless you have semantic exceptions enabled prior to checking your design:

```
entity E is
end;
library FOO; -- = WORK
entity FOO is
end;
architecture RTL of FOO is
  component E
  end component;
  for A : E use entity FOO.E;
-- in special mode, FOO after entity keyword is the library
-- and not the entity
begin
  A : E;
  B : entity FOO.E;
-- in special mode, FOO after entity keyword is the library
-- and not the entity
end;
```

VHDL Semantic Exception—Example 4

Do not hide component names using unit names. For example, the following code causes a compilation error unless you have semantic exceptions enabled prior to checking your design:

```
package P is
  component FOO end component;
end;
entity FOO is
end;
use WORK.P.all; -- makes component FOO potentially visible
use WORK.all; -- makes entity FOO potentially visible
entity TOP is
end;
architecture A of TOP is
begin
  I : FOO;
  -- in special mode, component FOO is visible
  -- (entity FOO is ignored)
end;
```

VHDL Semantic Exception—Example 5

Accept attribute specifications on entity declarative items in a corresponding architecture declarative part (illegal in VHDL 93 only). For example, the following code causes a compilation error unless you have semantic exceptions enabled prior to checking your design:

```
entity E is
  port (P : in BIT);
end;
architecture A of E is
  attribute FOO : INTEGER;
  attribute FOO of P : signal is 1;
  -- attribute specification must be written in entity
  -- declarative part; this is accepted in special mode
begin
end;
```

VHDL Semantic Exception—Example 6

Replace concatenation operations involving character literals and/or string literals using equivalent string literals. This allows you to use string literals in places where you require aggregates or locally static expressions, such as in subaggregate expressions or in case choices. For example, the following code causes a compilation error unless you have semantic exceptions enabled prior to checking your design:

```
architecture A of E is
    constant C : bit_vector(1 to 2) := "11";
    type M is array(1 to 2, 1 to 3) of bit;
    constant D : M := (("001"), ('0' & "10"));
    -- '0' & "10" accepted as sub-aggregate in special mode
begin
    process
        variable V : bit_vector(1 to 3);
        variable W : integer;
    begin
        case V is
            when "111" => W := 0;
            when '0' & C => W := 1; -- accepted in special mode
            when '1' & "01" => W := 2; -- accepted in special mode
            when others => null;
        end case;
        wait;
    end process;
end;
```

Writing & Checking Verilog Designs

You write Verilog code in modules or UDPs. There are two main standards in Verilog: one defined by the Verilog LRM, and another defined by industry-standard tools such as VCS. Both standards are accepted by the Checker. To ensure compatibility with the LRM, you must first compile or analyze the code. If compilation is successful, the internal representation of the compiled module or UDP is stored in a reserved location, usually a directory. The compiled modules can interrelate in a hierarchical way. In other words, modules can contain instantiations of other modules. You can store common resources, such as leaf cells, in a separate directory called a resource library. Leda refers to this combination of modules, UDPs, and resource libraries as a Verilog project.

If you want to apply chip-level checks, you must first elaborate the project or design. This means that you must resolve the connectivity between all the modules and UDPs in the design. If you instantiate a module in another module, the elaborator looks for the instantiated module and connects the ports as indicated by the instantiation. The Checker also applies a hardware inference algorithm to build an image used to validate hardware-based rules.

Verilog Semantic Exceptions

There are a number of semantic exceptions in Verilog. By default, the Checker observes these semantic exceptions. You can deselect this setting using the Project Update Wizard that comes up when you choose **Project > Edit** from the Checker's main menu. Click on the Verilog tab in this window, and deselect the "With semantic exceptions" check box button at the top left. Following are several Verilog code examples that cause compilation errors unless you have "With semantic exceptions" enabled prior to running the Checker.

Verilog Semantic Exception—Example 1

Accept the following non-standard compilation directives (LRM section 16) in exception mode. These all cause compilation errors unless you have "With semantic exceptions" enabled prior to running the Checker:

- 'accelerate
- 'autoexpand_vectornets
- 'default_decay_time
- 'default_rsswitch_strength
- 'default_switch_strength
- 'default_trireg_strength
- 'delay_mode_distributed
- 'delay_mode_path
- 'delay_mode_unit
- 'delay_mode_zero
- 'disable_portfaults
- 'enable_portfaults
- 'endprotect
- 'expand_vectornets
- 'noaccelerate
- 'noexpand_vectornets
- 'noremove_gatenames
- 'noremove_netnames
- 'nosuppress_faults

- 'protect
- 'remove_gatenames
- 'remove_netnames'
- 'suppress_faults

Verilog Semantic Exception—Example 2

Range definitions in a parameter declaration are defined in section 3.10 of the Verilog LRM. This standard does not allow parameters declared with a range, as shown in the following example:

```
parameter [3:0] p=4'b0101; // illegal in IEEE mode but legal in
                          // exception mode
```

This code causes compiler errors unless you have “With semantic exceptions” enabled prior to running the Checker.

Verilog Semantic Exception—Example 3

Assignment of the return value of a function is defined in section 10.3.4 e of the Verilog LRM. The LRM requires that in every function, an assignment to the return value is made (implicit variable with the same name as the function). The following code causes compilation errors unless you have “With semantic exceptions” enabled prior to running the Checker:

```
function foo;
foo = 4; // This line is mandatory in IEEE mode, not in the exception
        // mode
endfunction
```

Verilog Semantic Exception—Example 4

In a primitive declaration, when all the input values are set to X, the output must be X, according to section 8.1.4 of the Verilog LRM. Primitive declarations that violate this standard cause compilation errors unless you have “With semantic exceptions” enabled prior to running the Checker.

Verilog Semantic Exception—Example 5

In unsized based literals, when the size of the literal is less than 32, the size is set to 32 and the literal is padded on the left.

Writing & Checking Mixed-Language Designs

You can also check a project that contains both VHDL and Verilog source files. The semantic exceptions defined in the previous sections also apply to mixed projects. This section explains how to instantiate a block written in one language in the other language. Remember that instantiations are only resolved when you elaborate a project.

Instantiating a Verilog Module in a VHDL Architecture

You instantiate a Verilog module or UDP inside a VHDL architecture using a component instantiation statement. The instantiation of a Verilog module works just like an entity instantiation in a VHDL-only design: Note the following:

- When the default configuration mechanism applies, the Checker looks for an entity that has the same name as the component. If there is no entity, it looks for a Verilog module.
- When you explicitly specify a configuration and the entity does not exist, the Checker looks for a Verilog module.

Note that the default configuration must respect the usual rule in VHDL: same names in interface between VHDL component declaration and Verilog unit. If one port or parameter of the Verilog unit is not in upper-case or in lower-case, then you should use extended characters to avoid problems with casing conventions.

Instantiating a VHDL Design Entity in a Verilog Module

You can instantiate a VHDL entity, VHDL design entity, or VHDL configuration in a Verilog module using a module instantiation statement.

You can refer to this instantiated unit using either the name of the entity (as if it were a module), or by using a Verilog extended identifier to specify the configuration. [Table 4](#) provides some examples for instantiating a VHDL design entity in a Verilog module.

Table 4: VHDL Design Entity Instantiations in Verilog Modules

Example	Description
<pre>entity u1 (a, b, c);</pre>	This instantiates a VHDL entity named u1 that is located in the same library as the instantiating module. The architecture you use is the last compiled architecture of the entity.

Table 4: VHDL Design Entity Instantiations in Verilog Modules (Continued)

Example	Description
<code>\entity(arch) u1 (a, b, c);</code>	This instantiates a VHDL entity named u1 that is located in the same library as the instantiating module. The architecture you use is called arch. Note the use of extended identifiers to specify the architecture name.
<code>\MYLIB.entity u1 (a, b, c);</code>	This instantiates a VHDL entity named u1 that is located in a library called MYLIB. The architecture you use is the last compiled architecture of the entity. Note the use of extended identifiers to specify the library name.
<code>\MYLIB.entity(arch) u1 (a, b, c);</code>	This instantiates a VHDL entity named u1 that is located in a library called MYLIB. The architecture you use is called arch. Note the use of extended identifiers to specify the library name.

Mapping Data Types

In a mixed-language instantiation, you can map the VHDL ports to Verilog ports and ensure type equivalence as shown in [Table 5](#).

Table 5: Mapping VHDL Ports to Verilog Ports

VHDL Type	Verilog Type
integer	integer or real
real	integer or real
time	integer or real
physical	integer or real
enumeration	integer or real

Note the following:

- When a scalar type receives a real value, Leda converts the real to an integer by truncating the decimal portion.
- Treat type time as a special case. Leda converts the Verilog number to a time value according to the 'timescale directive of the module.

- Leda assigns physical and enumeration types with values that corresponds to the position number indicated by the Verilog number.

In a mixed-language instantiation, you can map the Verilog ports to VHDL ports and ensure type equivalence as shown in [Table 6](#).

Table 6: Mapping Verilog Ports to VHDL Ports

Verilog Type	VHDL Type
integer	integer
real	real
string	string

Leda also allows for the following other types in mappings:

- bit
- bit_vector
- std_logic
- std_logic_vector

Leda maps VHDL bit types to Verilog states as shown in [Table 7](#).

Table 7: Mapping VHDL bit Types to Verilog States

VHDL bit	Verilog State
'0'	St0
'1'	St1

Leda maps VHDL std_logic types to Verilog states as shown in [Table 8](#).

Table 8: Mapping VHDL std_logic Types to Verilog States

VHDL std_logic	Verilog State
'U'	StX
'X'	StX
'0'	St0
'1'	St1
'Z'	HiZ

Table 8: Mapping VHDL std_logic Types to Verilog States (Continued)

VHDL std_logic	Verilog State
'W'	PuX
'L'	Pu0
'H'	Pu1
'Z'	StX

Leda maps Verilog states to VHDL std_logic and bit types as shown in [Table 9](#).

Table 9: Mapping Verilog States to VHDL std_logic and bit Types

Verilog State	VHDL std_logic	VHDL bit
HiZ	'Z'	'0'
Sm0	'L'	'0'
Sm1	'H'	'1'
SmX	'W'	'0'
Me0	'L'	'0'
Me1	'H'	'1'
MeX	'W'	'0'
We0	'L'	'0'
We1	'H'	'1'
WeX	'W'	'0'
La0	'L'	'0'
La1	'H'	'1'
LaX	'W'	'0'
Pu0	'L'	'0'
Pu1	'H'	'1'
PuX	'W'	'0'
St0	'0'	'0'
St1	'1'	'1'

Table 9: Mapping Verilog States to VHDL std_logic and bit Types

Verilog State	VHDL std_logic	VHDL bit
StX	'X'	'0'
Su0	'0'	'0'
Su1	'1'	'1'
SuX	'X'	'0'

Leda maps Verilog states with ambiguous strengths as follows:

- bit receives '0'
- std_logic receives 'X' if either the 0 or 1 strength components are greater than or equal to strong strength.
- std_logic receives 'W' if both the 0 and 1 strength components are less than strong strength.

VHDL and Verilog Identifiers

Since VHDL is not case-sensitive and Verilog is case-sensitive, you must resolve names in mixed-language designs as explained in this section. The way each language stores identifiers remains unchanged. In other words:

- VHDL stores identifiers in upper case.
- Verilog stores identifiers exactly as they appear in the source code, regardless of case.

Verilog Instantiations of VHDL Design Units

There is no problem in this case. Leda looks for the instantiated module as it is written in the source code. If the module is not found, Leda assumes that it is a VHDL unit and converts the instantiation name to upper case. Leda also handles extended names (that is, those beginning with a \ character).

VHDL Instantiation of Verilog Design Units

You can instantiate any Verilog unit, regardless of its name. However, if the name is not all upper-case or all lower-case, then you should use the extended characters in the configuration specification or for the name of the component. [Table 10](#) shows some examples.

Table 10: Mapping VHDL Identifiers to Verilog Identifiers

Verilog Identifier	VHDL Identifier
TOPMOD	TOPMOD
topmod	topmod
TopMod	\TopMod\
top_mod	top_mod
_topmod	_topmod\
\topmod	\topmod\



Note

If you instantiate a Verilog module in a VHDL architecture without extended characters, Leda searches first using all upper-case then all lower-case letters. If neither one is found, Leda assumes that the module is a black box.

Port Naming in Default Associations

Component ports and generics should have the same names as instantiated unit ports and generics when considering insensitive cases. But here again, the instantiation order is important.

Verilog 2001 Support

Leda supports Verilog 2001 (V2K) constructs for language compliance checks. Most V2K constructs are fully analyzed, whereas some are simply parsed and ignored. Note that Leda accepts multidimensional arrays and references to them, but does not synthesize them.

Leda does not support Configurations. Also, the following Verilog 2001 features are all out-of-scope for Leda:

- Extended number of open files
- Enhanced file I/O
- Enhanced invocation option testing
- Enhanced SDF file support
- Enhanced VCD files
- Enhanced PLA system tasks
- Enhanced Verilog PLI support

To make Leda accept Verilog 2001 constructs in your source code when checking your designs for errors, add the `+v2k` switch to your Checker command-line invocation. This capability is not activated by default. Note that the `+v2k` switch is the same one used with the Synopsys VCS simulator for Verilog 2001 coverage. If you are using the GUI, you can make Leda accept Verilog 2001 constructs by selecting the 2001 radio button in the Project Creation Wizard (**Project** > **New** from the Specifier GUI main window). The default is Verilog 95.

SystemVerilog Support

Leda supports all of SystemVerilog 3.0, and SystemVerilog 3.1a assertions. To make Leda accept SystemVerilog constructs in your source code when checking your designs for errors, add the `+sv` switch to your Checker command-line invocation. This capability is not activated by default. Leda also accepts the `-sverilog` switch for compatibility with the Synopsys VCS simulator for SystemVerilog 3.0 coverage. If you are using the GUI, you can make Leda accept SystemVerilog constructs by selecting the SystemVerilog radio button in the Project Creation Wizard (**Project** > **New**) from the Specifier GUI main window). The default is Verilog 95.

Clock Grouping Feature

The clock grouping feature allows you to specify groups of synchronous clocks. The Clock Domain Crossing (CDC) rules take this information as inputs to avoid reporting violations on synchronous clocks.

To use the clock grouping feature do the following:

- Run Leda to extract the number of clocks from the design. If Leda detects more than the maximum number of clocks, it dumps the clock file and exits before the [chip-level, netlist and SDC checks](#) are run. You can set the maximum number of clocks using the environment variable LEDA_MAX_CLOCKS. The default value is 500. Set this variable as shown in the following example:

```
% setenv LEDA_MAX_CLOCKS 300
```

- Modify the clock file, using the clock grouping command `set_clock_groups`.
- Enable the clock file by setting the environment variable LEDA_CLOCK_FILE. Set this variable as shown in the following example:

```
% setenv LEDA_CLOCK_FILE ./TEST/clk_file.txt
```

You can set LEDA_CLOCK_FILE with a relative pathname or an absolute pathname and Leda accepts any extension for this file. However, Leda understands this file as a Tcl source file and uses it to set up user clock groups.

- Run Leda

You can use the following new variables (see [Table 11](#)).

Table 11: Environment Variables in Clock Grouping Feature

Label	Usage
LEDA_MAX_CLOCKS	Defines the maximum clock limit. Default value is 500.
LEDA_CLOCK_FILE	Sets this variable to the dumped modified clock file. The CDC rules take this information as input.

You can specify the clock groups using the following command:

set_clock_groups

Use the `set_clock_groups` command to specify exclusive or asynchronous clock groups.

Syntax

```
set_clock_groups -group clocks_list -asynchronous [-name name]
```

Arguments

-group	Specifies the list of clocks.
-asynchronous	Specifies the asynchronous clock groups.
-name	Specifies the name for clock grouping.

Example

```
set_clock_groups -name GR1 -group { top.clk1 top.clk2 } -asynchronous
```



Note

The clock grouping feature is not compliant with PrimeTime. It does not support options like multiple -group, -exclusive, and clock group removal.

- You can use the command line option -clock_file to specify the synchronous clocks in the design through the set_clock_groups command. The checker uses this information for doing chip-level, netlist and SDC checks. You should specify the file name (leda_<topname>_clock.tcl) with the -clock_file option. For example:

```
% leda -top topunit test.v -clock_file leda_topunit_clock.tcl -config  
cfg.tcl
```



Note

Using the environment variable LEDA_CLOCK_FILE to specify the clock file is equivalent to using the -clock_file command line option.

Netlist Reader

The Netlist Reader is an optimized Verilog compiler for reading large netlists. It performs optimizations on the intermediate database generated, to ensure that only relevant information is present. The netlist reader has the capability to read any Verilog netlist faster, and with much less memory consumption. This enhances performance to a large extent. The netlist reader accepts only a minimal subset of language constructs. You cannot run block-level checks on code that is read by the Netlist Reader. To ensure this, the block-level checker is automatically disabled when the netlist reader is invoked. If your input Verilog netlist file contains syntax that is not recognized by the netlist reader, then the standard Verilog compiler is automatically invoked.



Note

If you are using the Netlist Reader, you should run only the Chip-level and Netlist-level rules.

Invoking the Netlist Reader

You can invoke the Netlist Reader from the command line, the Tcl shell, or the GUI. When you use the Netlist Reader, it automatically disables testing for leda on/off macros and disables the block-level checker.

From the command line, invoke the netlist reader by using the option `-use_netlist_reader`. For more information, refer [“Common Command-Line Options and Switches” on page 148](#).

From the Tcl shell, invoke the netlist reader using the option `-netlist_reader` with the following read commands.

```
leda> read_verilog -netlist_reader
leda> read_sverilog -netlist_reader
leda> read_files -format verilog -netlist_reader
```

For more information, refer [“Rule Tcl Command Reference” on page 197](#).

In GUI mode, invoke the netlist reader by checking the “Activate Netlist Reader” checkbox in the “Project Creation Wizard” window under the section “Specify Compiler Options”.

You can run the netlist reader only on a single file. It ignores preprocessing options such as `-y`, `-v`, `+incdir`, etc.

Netlist Reader BNF

This section describes the BNF grammar accepted by the netlist reader. If any other constructs are present in an input file, the netlist reader treats it as a syntax error and exits. Then the standard Verilog compiler attempts to compile the file.

```

source_file ::= { module_definition }
module_definition ::= module identifier [ ( list_of_ports ) ] ;
    module_item_declarations
    module_statements
endmodule
list_of_ports ::= port { , port }
port ::= identifier

module_item_declarations ::= { net_type [ range ]
    list_of_net_identifiers ; }
net_type ::=    wire
    | tri
    | input
    | output
    | inout
    | supply0
    | supply1

range ::= [ integer : integer ]
list_of_net_identifiers ::= identifier { , identifier }

module_statements ::= { assign_statement
    | instantiate_statement
    }

assign_statement ::= assign identifier =
    name_id
    | numeric_value
    ;

```

```
instantiate_statement ::= identifier [identifier]
                        (list_of_port_connections ) ;
list_of_port_connections ::= port_connection { , port_connection }

port_connection ::= . identifier ( actual )
                 | actual

actual ::= name_id
         | concatenation
         | numeric_value

name_id ::= identifier [ [ integer [: integer ] ] ]
numeric_value ::= 1'b1 | 1'b0 | 1 | 0
```

3

Modifying and Creating Rules

Introduction

There are two main tools in Leda: the Specifier and the Checker. You use the Specifier to build rules that are then used by the Checker on your HDL code to ensure that it complies with those rules. You need an optional Specifier license to create and compile new rules, but a Checker license is all you need to configure the prepackaged rules.

This chapter provides detailed procedures for how to configure the prepackaged rules and create new rules to check your HDL code with, in the following major sections:

- [“About Rules, Rulesets, and Policies” on page 72](#)
- [“Using Configurations” on page 72](#)
- [“Configuring the Rule Wizard” on page 73](#)
- [“Configuring Prepackaged Rules” on page 74](#)
- [“Locking the Rule Wizard” on page 75](#)
- [“Using the Rule Wizard to Configure Rules” on page 77](#)
- [“Creating New Rules” on page 79](#)
- [“Defining Macro Values for Rules” on page 82](#)
- [“Exporting and Importing Policies” on page 86](#)

About Rules, Rulesets, and Policies

Leda organizes rules into rulesets. Rulesets are stored as ASCII text in *ruleset.rl* files for VHDL or *ruleset.sl* files for Verilog. A ruleset can contain any number of rules and template declarations, as well as previously compiled templatesets. A templateset is similar to a ruleset, except that it only contains template declarations. Note that if a ruleset uses a templateset and this templateset is recompiled, the ruleset becomes obsolete and must also be recompiled. This is analogous to a VHDL architecture becoming obsolete if its entity is recompiled.

A policy can contain any number of rulesets. If you are creating new rules yourself, you can organize your rulesets into different policies in the way that is easiest for you to manage. Each policy is represented as a top-level item in the rule hierarchy shown in the Rule Wizard. You use the Rule Wizard to activate or deactivate rules for checking your HDL code (see [“Using the Rule Wizard to Select or Deselect Rules” on page 98](#)).

Leda compiles rules from ASCII-based source code in *ruleset.rl* or *ruleset.sl* files and places the results in policy libraries, which are stored in the `$LEDA_PATH/.leda_config.files` directory. This is the default configuration.

Using Configurations

In Leda, a configuration is any file that you specify with `$LEDA_CONFIG` in the shell or using the `-config` option in batch or Tcl modes. You usually create configuration files using the Rule Wizard, but you can also write them by hand once you know the syntax (see [“Rule Tcl Command Reference” on page 197](#)). A configuration file must be an ASCII text file that contains valid Tcl commands for Leda.

In addition, a configuration may include a *directory.files* directory that contains compiled rule binaries for custom rules that you create with the Policy Manager using the Specifier tool. For example, if you have `$LEDA_CONFIG` pointing to a local or custom rule installation, Leda stores the policies in the `$LEDA_CONFIG.files` directory. If you have `$LEDA_CONFIG` set to:

```
/home/fr03/julius/FOO
```

and you used the Rule Wizard in the Specifier to create a new policy, in the toolbar Leda says that the current configuration is saved into `/home/fr03/julius/FOO`. In this case, Leda saves your compiled rule binaries in a directory named `/home/fr03/julius/FOO.files` and your configuration commands in a file named `/home/fr03/julius/FOO.tcl`. So, a configuration consists of a configuration file, and in the case of custom rules, a *directory.files* directory that contains the compiled custom rules.

Configuring the Rule Wizard

If you have a Checker-only license, the first time you use the Rule Wizard to configure prepackaged rules, Leda loads the default configuration located in the `$LEDA_PATH/.leda_config.tcl` file. The default configuration contains about 70 prepackaged rules for RTL checks. This default RTL configuration is one of four prebuilt configurations that you can use with Leda (see [“Using Prebuilt Configurations” on page 99](#)).

To use a configuration other than the default (`$LEDA_PATH/.leda_config.tcl`), point to your configuration file using the `$LEDA_CONFIG` variable in the shell or use the `-config` option in batch or Tcl modes. For example:

```
% setenv LEDA_CONFIG /u/julius/leda/my_config.tcl
```

You can name your configuration file with any file name and extension that you want. However, it is good practice to name configuration files with a `.tcl` extension for ready identification. Your configuration file must contain valid Tcl commands for Leda (see [“Rule Tcl Command Reference” on page 197](#)).



Caution

If you are a Checker-only user, do not set the `LEDA_CONFIG` variable pointing to an empty directory before invoking Leda for the first time. This causes Leda to issue a warning message about not being able to find the policies that contain prepackaged rules. If you run into this, exit the tool, unset the `$LEDA_CONFIG` variable, and then re-invoke Leda.

The last configuration that you save when working with the Rule Wizard becomes the default configuration for the current session.

Saving Configurations

To save a configuration, choose **Config > Save As** from the Rule Wizard window, and navigate to a directory where you have write permissions. In the Save Current Configuration As window, specify the configuration file. Leda saves your configuration in a `config.tcl` configuration file in that directory. If you have custom rules in your configuration, Leda also saves the binaries for those rules in a `directory.files` directory there.

Restoring Configurations

Leda does not have a restore capability for configurations that you work with in the tool. If you want to be able to go back to the configuration you started with for any reason after using the Rule Wizard or Tcl shell commands to configure rules, first save a copy of the configuration file pointed to by `$LEDA_CONFIG` and the `$LEDA_CONFIG.files`

directory (if present) someplace where they will not get overwritten. Then if you want to restore your configuration to the same one you started with, exit the tool and copy the saved configuration file and \$LEDA_CONFIG.files directory (if present) back to the location pointed to by \$LEDA_CONFIG and restart the tool.

Rule Configuration Search Path

In all cases, Leda references the first configuration file it finds in the following list. Leda also uses this search path when you save changes you make to prepackaged rules using the Rule Wizard:

- -config batch option (see “-config full_path_to_file” on page 149)
- \$LEDA_CONFIG
- \$cwd/.leda_config.tcl
- \$HOME/.leda_config.tcl
- \$LEDA_PATH/.leda_config.tcl (default configuration)

Global Checking with the Same Rule Configuration

Another option is to set one configuration for the prepackaged rules that is referenced by all engineers at your site. This can be useful for managers who want to make sure that all engineers are checking their HDL designs using the prepackaged rules in exactly the same way to ensure consistency. To configure the Rule Wizard this way, first set the LEDA_CONFIG environment variable to a global rule configuration file as follows:

```
% setenv LEDA_CONFIG path_to_global_rule_configuration_file
```

Then, invoke Leda and make your changes to the prepackaged rules, as explained in “Using the Rule Wizard to Configure Rules” on page 77. To make your changes apply globally, have all engineers checking rules at your site set the LEDA_CONFIG environment variable in their shell sessions to the *global_rule_configuration_file* you set up.

Configuring Prepackaged Rules

If you just want to change the naming conventions used in a prepackaged rule or specify a check to occur on the rising or falling edge of the CLK, for example, you can use the Rule Wizard to configure the existing rule to meet your needs. All you do is modify the value argument or node for a rule to the setting you want and save your changes. (Note that not all of the prepackaged rules have value arguments.) You can also change the rule label, error message, and message severity for any of the prepackaged rules.

For details about the current set of prepackaged rules available with Leda, see the [Leda Prepackaged Rules Guide](#). It is a good idea to familiarize yourself with what is available in the prepackaged rules before creating new rules yourself.

Locking the Rule Wizard

If you have a Specifier license and write permissions to \$LEDA_PATH, you can lock the Wizard so that other users at your site cannot modify prepackaged or custom rule configurations. Follow these steps:

1. Make sure \$LEDA_CONFIG is not set to a custom rule configuration. The Rule Wizard must be pointing to the default rule configuration in \$LEDA_PATH/.leda_config.tcl in order to lock the Wizard for all rule configurations which reference the Leda installation in \$LEDA_PATH:

```
% unsetenv LEDA_CONFIG
```

2. Invoke the Leda Specifier GUI:

```
% $LEDA_PATH/bin/leda -specifier &
```

3. From the Specifier's main window, choose **Check > Configure**. This brings up the Rule Wizard.
4. From the Rule Wizard window, choose **Config > Checker Controls**. This brings up the Checker Control Panel (see [Figure 12](#)).



Figure 12: Checker Control Panel

5. The “Activate rule config in Rule Wizard” checkbox in the Permissions panel is selected by default. To lock the Wizard, deselect this checkbox so that it appears as shown in [Figure 12](#), and click the Close button.

With the Wizard locked, rules in the default configuration and any custom rule configurations cannot be changed by other users. Note that when the Wizard is locked, Leda ignores rule_select and rule_deselect Tcl commands.

When a user invokes the locked Rule Wizard (**Check > Configure**), Leda issues a warning message that explains the current locked status (see [Figure 13](#)) before bringing up the Rule Wizard window. If the Wizard is locked and you need to make changes, see your system administrator.

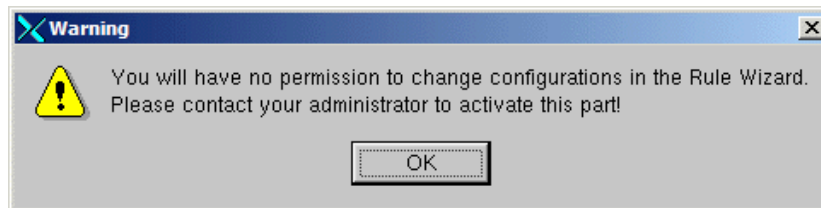


Figure 13: Locked Rule Wizard Warning

Using the Rule Wizard to Configure Rules

To run the Rule Wizard, choose **Check > Configure** from the Specifier or Checker main window. This brings up the Rule Wizard window (see [Figure 14](#)). Note that the tool displays the configuration that it loaded in the message area near the top-left corner. To load a different rule configuration, choose **Config > Load configuration**, and then **Custom**. Use the resulting “Load a Configuration” window to navigate to the directory where your configuration file is located.

Configuration Loaded

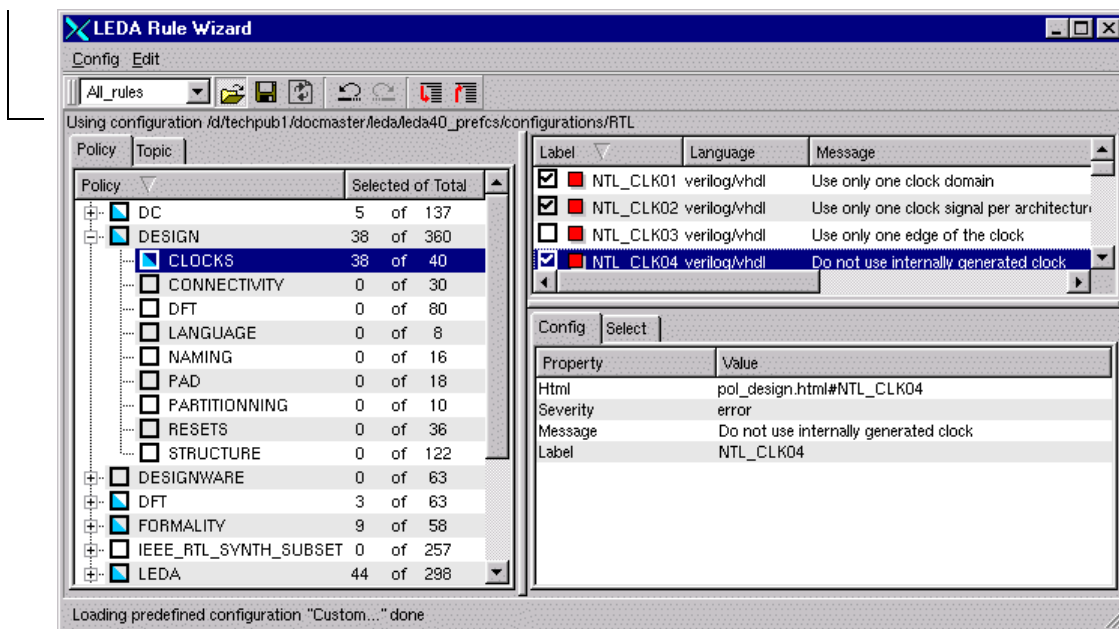


Figure 14: Rule Wizard Window

Policy and Topic Views





The Rule Wizard has several tabs and panels. The Topic tab on the left side lists rule topics in general categories that span multiple policies (for example, Clocks). The Policy tab shows you the policies that contain these rules. The two tabs provide different views of the same database of prepackaged rules. The top panel on the right side is blank until you either select a ruleset from within a policy in the Policy tab or from the general categories in the Topic tab. Then the top panel on the right fills up with all the rules from the selected ruleset. Click on the Label, Language, and Message bars to sort the display on any of these items in ascending or descending order. To deselect a rule for checking, click on the check box. When you click on another rule, the check box appears blank,

confirming that the rule is now deselected for checking. To select a rule for checking, click on the blank check box. When you click on another rule, the check box appears with a check mark inside, confirming that the rule is now selected for checking.

Configuring Rule Properties

When you click on an individual rule in the top-right window, the Config tab on the lower right of the Rule Wizard displays the Severity, Message, Label, and HTML help file name. To change the Severity for a rule, click on the Severity line to reveal a menu from which you can choose a new setting. For the other configurable items, double-click on the line and make your changes. Each rule in the tree has a colored button next to the selection check box that shows the severity level for that rule (see [Table 12](#)).

Table 12: Rule Severity in Rule Wizard

Rule Color	Severity
	Note
	Warning
	Error
	Fatal



Caution

For the best general-purpose results with prepackaged rules, it is advisable to leave the rule labels and HTML help file names at their default values. In particular, if you change the name of the HTML help file for any of the prepackaged rules, the HTML-based help system will not work.

To save your changes, choose **Save** or **Save As** from the Config menu. The save option saves the configuration in the directory indicated near the top left corner of the window (using configuration...). If you want to save the configuration elsewhere, use the Save As option and choose a directory where you have write permissions. Once saved, this configuration becomes the current loaded configuration. If you want to use a different configuration, choose **Config > Load configuration**, choose Custom from the pull-down menu, and navigate to the desired configuration file.

Creating New Rules

If the current set of prepackaged rules does not meet all of your HDL checking needs, you can either copy and modify an existing rule or write a new rule from scratch. This section explains how to create new rules using both methods. Before proceeding, review [Table 13](#) to determine which approach best meets your needs.

Table 13: Choosing a Method for Creating New Rules

If ...	Then ...
One of the rules listed in the <i>Leda Prepackaged Rules Guide</i> is similar to the rule you want to implement.	Copy and modify the rule source code from the \$LEDA_PATH/rules/policy directory, as explained in “Copying and Modifying Prepackaged Coding Rules” on page 80. This is the easy way. It should cover a high percentage of your custom rule needs. The advantage here is that it is faster and you don’t need to learn much about the rule specification languages to get what you want.
None of the rules listed in the <i>Leda Prepackaged Rules Guide</i> is similar to the rule you want to implement.	Write your own custom rule from scratch using VRSL (for VHDL) or VeRSL (for Verilog), as explained in “Writing New Rules from Scratch” on page 80. The advantage here is that you develop expertise with the rule specification languages that you can later apply to future rule-creation needs.



Note

You can also write design netlist-checking rules using Leda’s Tcl or C rule APIs. You integrate compiled rule (C) or Tcl scripts (.tcl) into the Leda environment using VeRSL wrappers and ruleset files, just like coding rules (see [“Creating New Policies”](#) on page 81). For more information, see the *Leda Tcl Interface Guide* or *Leda C Interface Guide*.

Copying and Modifying Prepackaged Coding Rules

To create a new coding rule using the copy-and-modify method, first review the *Leda Prepackaged Rules Guides* to find one or more rules that are close to what you need. There is one PDF file for each policy. The PDF books are located in the \$LEDA_PATH/doc directory. Note the policy names and rule labels so that you'll be able to easily find the VRSL or VerSL source code in the Leda installation. Write down the rule labels of interest and then follow these steps:

1. Navigate to the \$LEDA_PATH/rules/*policy* directory, where *policy* is one of the current prepackaged policies. For example:

```
% cd $LEDA_PATH/rules/rmm
```

2. List out the directory contents and note the .rl and .sl files. These are ruleset files that contain the rule source code. For VHDL, open the applicable .rl ruleset file. For Verilog, open the applicable .sl file.
3. Use your text editor to create a new *my_ruleset.rl* file for VHDL or *my_ruleset.sl* file for Verilog.
4. For each rule label, search for that rule's source code in the .rl file (VHDL) or .sl file (Verilog). Copy all of the source code for each rule to your new *my_ruleset.rl* or *my_ruleset.sl* file. Be sure to copy both the "command" and "template" sections for each rule.
5. Modify the rule source code as needed, and save the file.
6. Now that you have a customized ruleset file, you can proceed to the section on "[Creating New Policies](#)" on page 81 and take it up from there.



Note

To learn how to modify VRSL or VerSL code to meet your needs, see the [Leda Rule Specifier Tutorial](#). For complete syntax and reference information, see the [VRSL Reference Guide](#) (VHDL) and [VerSL Reference Guide](#) (Verilog).

Writing New Rules from Scratch

To write new rules from scratch, first invoke the Specifier tool as follows (you use the same tool for both Verilog and VHDL):

```
% $LEDA_PATH/bin/leda -specifier &
```

The Specifier main window opens; it looks identical to the Checker main window. The only difference is the presence of a Policy Manager window, which you access through the Rule Wizard (**Check > Configure**, then **Tool > Policy Manager**).

Creating New Ruleset Files

To create new ruleset files, follow these steps:

1. Using a text editor, type in the VRSL or VerSL code for the new rules that you want to create. You can use the text editor in the Specifier by choosing **File > New**. For information on selecting an editor other than the Leda default text editor with Leda, see [“Selecting a Text Editor” on page 172](#).
2. For VHDL, create the file as *ruleset.rl*. For Verilog, create the file as *ruleset.sl*. Note that “.rl” is the standard extension for VHDL ruleset files and “.sl” is the standard extension for Verilog ruleset files.




Hint

You can find example VRSL and VerSL code in the `$LEDA_PATH/doc/tutorial_specifier/rsl` directory. The file names are *ruleset.rl* (VHDL) and *ruleset.sl* (Verilog).

Creating New Policies

Once you have a new *ruleset.rl* or *ruleset.sl* file that you either created by copying and modifying some of the prepackaged rules, or by writing the rules from scratch, you need to store the new rules in a policy. To do that, follow these steps:

From the Specifier main menu, choose **Check > Configure**. This opens the Rule Wizard.

From the Rule Wizard, choose **Tool > Policy Manager** or click on the  button (see [Figure 15](#)). Note that the Specifier and Checker GUIs look identical, except that only the Specifier tool includes the policy manager (`$LEDA_PATH/bin/leda -specifier`).

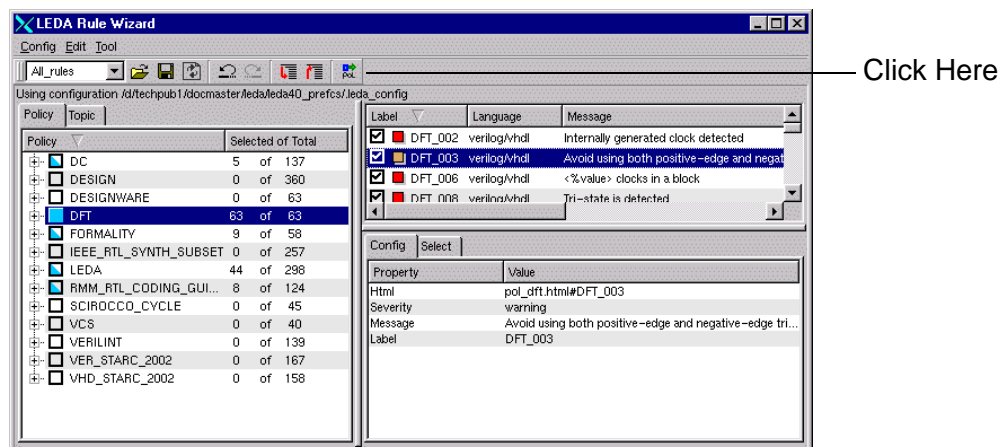


Figure 15: Invoking the Policy Manager

1. From the Policy Manager window, click on the VHDL or Verilog tab, as applicable.
2. Click on the New button on the right side of the display. Type in a name for the new policy (for example, “my_policy”) and click on OK.
3. When the new policy appears in the left pane, click on it to highlight the name and then click in the Rulesets pane. Click on the Add button.
4. Navigate to the location of the ruleset file you just created and click on the file name (*ruleset.rl* or *ruleset.sl*). Then click on the Add button. This causes the tool to compile your new rulesets.
5. Close the Policy Manager window.

You have now created a policy containing the rules defined in the *ruleset.rl* or *ruleset.sl* file you developed. Your new policy and the rules it contains now appear in the Specifier main window. Now you can select them for checking. You can later add any number of new ruleset files to the same policy depending on how you want to organize your custom rules.

Defining Macro Values for Rules

The Verilog and VHDL rule specification languages (VeRSL and VRSL, respectively) allow for the use of macros that you later define for rule checking using a configuration file that you put in Leda’s [Rule Configuration Search Path](#) (see [page 74](#)). In case of conflicts, Leda uses the macros found in the last file found in this path.

Defining macro values for rules may be important because, in some cases, rules may be too strict. For example, when testing legacy code, you may violate some naming convention rules many times. You could always switch off these rules, but it is also useful to be able to modify the rules online. For example, in the RMM policy, clock signals must have the prefix `clk_`. This rule is written as follows:

```
template CLOCK_ID is clock
    limit identifier to "^clk_"
end
```

This rule is often violated in code that was written before the RMM was created or that used another naming convention for clock signals. You can extend the rule definition to include a variable parameter as follows:

```
template CLOCK_ID is clock
    limit identifier to "<clock_name>"
end
```

If you modify a rule this way, you must then pass a value to the parameter `clock_name` (the default is the name of the macro itself) before checking the rules by defining the value for the `clock_name` macro in a configuration file using the following Tcl command:

```
rule_set_parameter -rule rule_label -parameter (label | macro_name) \  
-value value
```

For example, to set the value for the `clock_name` macro, use the following command:

```
rule_set_parameter -rule B_4404 -parameter clock_name -value clk
```

You can use simple regular expressions to define macro values. For example, to define the `clock_name` parameter used in a rule, put the following line in your configuration file:

```
rule_set_parameter -rule B_4404 -parameter clock_name -value \  
{^clk_|^clock$|^clk_n$|^clk_r$}
```

This loosens up the rule for clock naming, so that all the cases covered in this macro definition become legal and do not cause the Checker to flag an error.



Note

In general, results you get using regular expressions with Leda are the same as you get using `grep`. Both programs use `regex (5)` to parse regular expressions. Note that Leda currently supports simple regular expressions, but not extended regular expressions of the form `\{m\}`, `\{m,\}`, or `\{m,n\}` that are supported with `egrep` or `grep -E`. Also, Leda uses the GNU version of `regex`, which differs slightly from the UNIX version (for details see the man pages).

For information on a set of predefined `rule_set_parameter` Tcl commands for prepackaged rules that you can cut-and-paste from the manual and modify as needed, see [“Predefined Macros for Prepackaged Rules” on page 243](#).

Using Predefined Macros to Constrain Identifiers

To allow for the representation of strings containing unit names, reserved strings are defined that you can use as part of other strings. For example, in VerSL, one such string is:

```
<module>
```

If the Checker encounters the above sequence of characters in a string, it replaces it with the name of the corresponding unit or module. You specify file names that include the name of a unit as follows:

```
limit file_name in module_declaration to "<module>.v"  
    message "Illegal file name for module"  
    severity error
```

If the Checker is testing a module M that is not in file M.v, you get an error. Note that you can also create your own user-defined macros. You could rewrite the above rule using a user-defined macro called `module_filename`, as shown in the following VerSL example:

```
limit file_name in module_declaration to "<module_filename>"  
    message "Illegal file name for module"  
    severity error
```

Then, you set the value of your macro in the configuration file using a Tcl command as follows:

```
rule_set_parameter -rule rule_label -parameter module_filename -value  
    <module>.v
```

For VHDL rules written in VRSL, you set values for the following variables in rules by defining these values in your configuration file:

- architecture
- entity
- configuration
- component
- package
- library
- type
- formal
- target

Advanced Macro Programming

You can also build macros based on other macros. For example, you can create a basic macro that defines a name and others that derive new names from the first one. If you want to constrain an attribute to a multiple choice of regular expressions, you can define a regular expression that expresses these multiple choices in the configuration file. This makes the rule easier to understand and maintain. You use the same Tcl command syntax, as shown in the following example:

```
set module_index ...
set module_more_info_index ...
set module_name01 module0; #module0 is a possible name for the instance
set module_name02 $module_index
set module_name03 $module_more_info_index

rule_set_parameter -rule rule_label -parameter instantiated_module_name
-value "$module_name01 | $module_name02 | $module_name03"
```

The definition order for macros in the configuration file is important. You must define a macro before using it. Otherwise you get a Tcl error.

Constraining Max/Min Attributes to Predefined Values

You can use the max and min commands to constrain the number of elements in a given Verilog clause. For example, suppose you want to limit the number of characters for the module names. You can write this rule as follows:

```
template AB is identifier
    max character_count is 20
end
template MOD is module_declaration
    limit identifier to AB
end
```

You can also use these commands to disable the use of initial constructs in modules, as shown in the following example:

```
max initial_construct in module_declatation is 0
```

In some cases, it may be useful to constrain a max/min attribute to a value you define in a macro. This makes the rule a generic one that you can use with your own parameters. Then, to change the value of the macro, you only need to modify the configuration file, while the original rule definition remains the same. For example, you could define the previous rules as follows:

```
template AB is identifier
    max character_count is "<max_length>"
end
template MOD is module_declaration
    limit identifier to AB
end
max initial_construct in module_declatation is "<my_value>"
```

And then set the values for the *max_length* and *my_value* macros in your configuration file as shown in the following examples:

```
rule_set_paramter -rule rule_label -paramater max_length -value 20
rule_set_parameter -rule rule_label -parameter my_value -value 0
```

Exporting and Importing Policies

At companies with multiple sites, you may need to build and modify policies at one site owning Specifier licenses, and then export those policies to another site that only has Checker licenses. Perhaps there is no direct network link between sites, or this link is too slow or costly.

Leda provides a utility called `export` that creates a tar file containing a binary version of a new policy. This executable is located in the `$LEDA_PATH/utilities/export` directory. To use it, enter the following command from a directory where you have write permissions:

```
% $LEDA_PATH/utilities/export/export policy_name
```

where *policy_name* is the name of the policy to be created.

For example:

```
% $LEDA_PATH/utilities/export/export IEEE_RTL_SYNTH_SUBSET
```

The `export` utility creates a tar file called `ieee_rtl_synth_subset_rules.tar` that you can send to other locations. There is a utility called `import_policy_name` included in the tar file that remote users can use to automatically re-install the exported policy. To install an updated policy, type the following:

```
% mv policy_name_rules.tar $LEDA_PATH/rules/vhdl
% tar -xvf policy_name_rules.tar
% import_policy_namecv
```

**Note**

You can export and import policies across platforms between Solaris and HP-UX, but there is no cross-platform compatibility between Linux and Solaris or HP-UX.

4

Checking Designs For Errors

Introduction

This chapter explains how to build project files for your HDL code, select custom or prepackaged rules to check it against, execute the Checker, and fix any errors that are found. This information is organized in the following major sections:

- [“Invoking the Checker GUI” on page 90](#)
- [“Creating Projects to Check HDL Code” on page 91](#)
- [“Propagating Constants” on page 96](#)
- [“Using the Rule Wizard to Select or Deselect Rules” on page 98](#)
- [“Deactivating Rules” on page 101](#)
- [“Setting & Saving Checker Preferences” on page 108](#)
- [“Running the Checker” on page 109](#)
- [“Fixing Errors Found by the Checker” on page 112](#)
- [“Sorting the Error Viewer Display” on page 116](#)
- [“Filtering the Error Viewer Display” on page 117](#)
- [“Viewing the Design Report” on page 120](#)
- [“Using the Path Viewer” on page 121](#)
- [“Using the Clock and Reset Tree Browsers” on page 126](#)
- [“Saving Error Reports” on page 127](#)
- [“Post-processing Batch Mode Log Files” on page 128](#)
- [“Updating Projects” on page 130](#)

You can perform all the tasks in this chapter using either the standalone Checker tool, or the Specifier tool, which is the same as the Checker except that it allows you to build and compile new rules. Because these tools are so similar, the GUI displays for them are almost identical. The only difference is that the Specifier has a Policy Manager window (**Check > Configure**, then **Tool > Policy Manager**) that is not present in the Checker tool. You use the Policy Manager window to compile new rules for the Checker.

Invoking the Checker GUI

First, make sure your environment is set up correctly (see [“Setting Leda Environment Variables”](#) on page 317). Then, invoke the Checker as shown in the following example:

```
% $LEDA_PATH/bin/leda &
```

This brings up the Checker main window (see [Figure 16](#)). All the menus and functions in the Checker tool are also available from the Specifier tool.

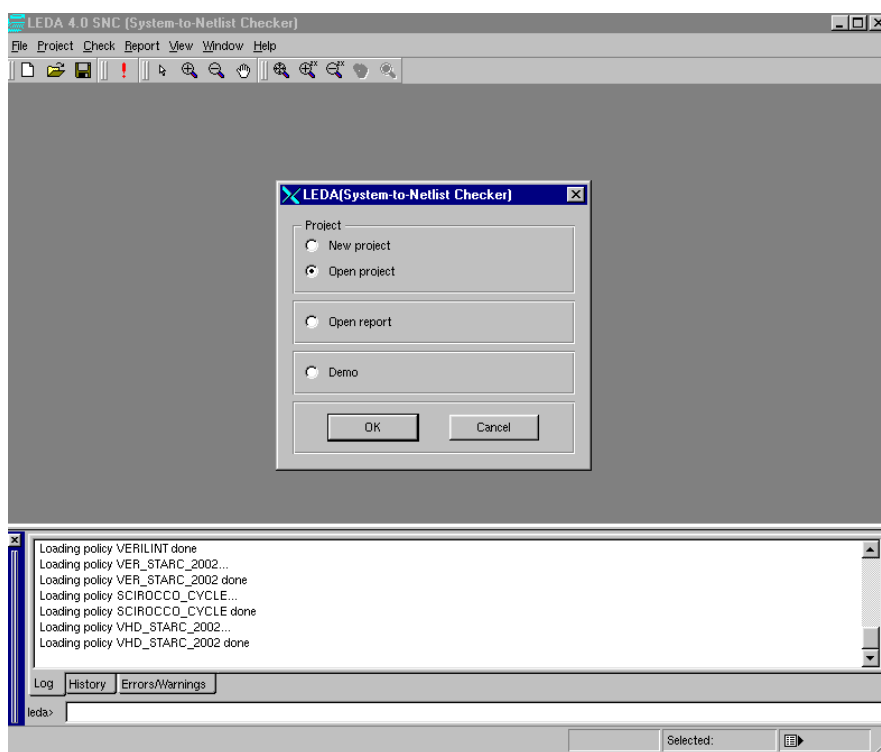


Figure 16: Leda Checker Main Window



Note

For information on checking the environment that Leda is currently referencing, see [“Checking Your Environment”](#) on page 171.

Creating Projects to Check HDL Code

Before you can use Leda to test your HDL source code against the prepackaged rules or new rules that you built, you must first create a project file. A project file organizes your VHDL or Verilog source files into easily managed units. Follow these steps:

1. From the main menu, choose **Project > New**. This opens the Project Creation Wizard window. Enter the full path and name for your project in the Project Name field at the top of the window, or enter just a project name and use the Browse button to navigate to a directory where you want to store your project data. Then click the Next button at the bottom right of the window to start the Wizard (see [Figure 17](#)).

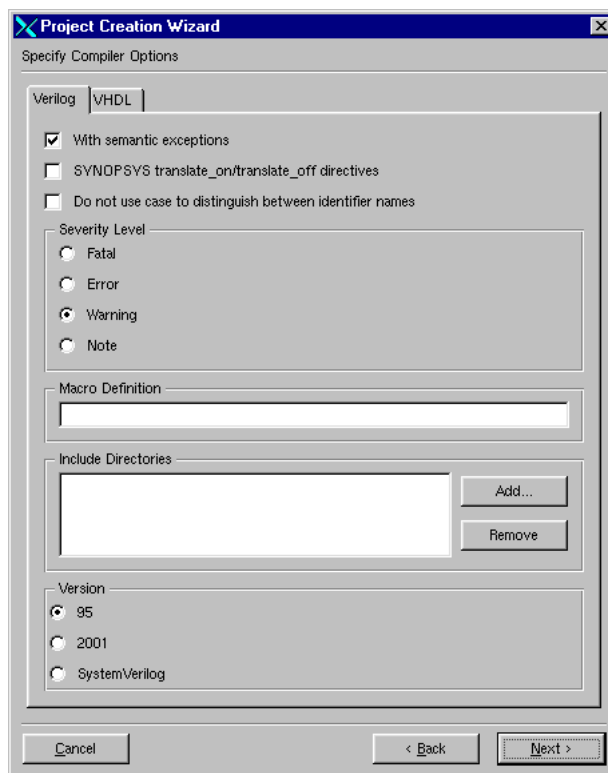


Figure 17: Project Creation Wizard Window

2. The first step in the Wizard is to Specify Compiler Options, as reflected in the title at the top of the display. The display has tabs for VHDL and Verilog. For both languages, there are check boxes that appear at the top of the tab:

With semantic exceptions

The first is “With semantic exceptions,” which is enabled by default. With this check box selected, Leda observes semantic exceptions when it analyzes your HDL source code. For more information on semantic exceptions, see “[VHDL Semantic Exceptions](#)” on page 52 and “[Verilog Semantic Exceptions](#)” on page 57.

SYNOPSYS translate_on and translate_off directives

The second is “SYNOPSYS translate_on translate_off directives,” which is disabled by default. If you want Leda to honor these pragmas in your HDL source code, select this check box. With this feature enabled, Leda does not attempt to translate code that is delimited by these pragmas. This can be useful for non-synthesizable code such as testbenches in your project, for example, that would cause unpredictable results for hardware-based rules if you leave this option disabled.

Do not use case to distinguish identifier names

This check box appears only for Verilog, since VHDL is not case-sensitive. The option is disabled by default. Select this check box if you want Leda to consider identifier names to be the same where the case differs.

3. In the Severity Level panel, click on the radio button for the lowest severity level for which error messages from the VHDL or Verilog analyzer will be printed. Analyzer messages with a severity below the specified value are not printed. (This severity level is only used for syntax analysis, not for checking.) The default is Warning.
4. Still from the Specify Compiler Options window in the Wizard, proceed to the language-specific setup tasks for VHDL and Verilog:

VHDL

- m In the Version pane, click on the 87 or 93 radio button, depending on the version of VHDL you are using. The default is VHDL 93. You cannot mix VHDL 97 and VHDL 93 in the same design and check it with Leda.

Verilog

- m In the Include Directories pane, use the Add button to navigate to any directories that you want to be searched for included files in your design. To remove an include file, select it in the window, and click the Remove button.
- m In the Macro Definition field, specify the values for any preprocessor macros that you want to be in effect for the analysis.

- m In the Version pane, click on the 95, 2001, or SystemVerilog radio button, depending on the version of Verilog you are using. The default is Verilog 95. For information on Leda's support levels for Verilog 2001 and System Verilog, see [“Verilog 2001 Support” on page 65](#) and [“SystemVerilog Support” on page 65](#).
5. When you are done specifying your compiler options, click on the Next button at the bottom of the window to proceed to the next step, Specify Libraries.
 6. The Specify Libraries window has tabs for VHDL and Verilog:

VHDL

- m In the Working Libraries pane, specify the logical names of working libraries where your VHDL analyzer will store binary results of the VHDL analysis. If you are not using any specific working libraries, leave this pane empty and the tool will put your analyzed code into the default location.
- m In the Resource Libraries pane, specify the logical names and mappings to the physical locations of additional existing compiled resource libraries. These are golden libraries that can be shared by multiple projects and users and usually contain common packages. (By default, the standard IEEE, STD, and Synopsys libraries are available.)

For more information on specifying and managing VHDL libraries, see [“Managing VHDL Libraries and Files” on page 313](#).

Verilog

- m In the Working libraries pane, click on the New button and use the New Library Window to specify the name of a working library that you want to add to your project. When you click OK, the new library name appears in the Working libraries pane. To remove a working library from the project, select the library name in the pane and click the Remove button.
 - m In the Library directories and Library files panes, click on the Add button and navigate to the locations of any required source code libraries to be searched by the Verilog analyzer in order to resolve unresolved module instances. Click on the Enable Checks check box if you want Leda to check selected rules in the specified library directories or files.
7. When you are done specifying your compiler options, click on the Next button at the bottom of the window to proceed to the next step, Specify Source Files.

8. The Specify Source Files window has tabs for VHDL, Verilog, and All. (The All tab is for mixed-language designs.) Source files, in this case, means VHDL or Verilog source files, the ones you want to check:

VHDL

- m In the Directories/Files pane, click on the Add button. This opens the Add Directory/File window. Navigate to the location of the .vhd or .vhdl files you want to check. Highlight the file names and click on the Add button. Then Click on the OK button to confirm your selections. You now see the full paths to the directories or files you specified displayed in the Directories or Files panes.

Verilog

- m In the Directories/Files pane, click on the Add button. This opens the Add Directory/File window. Navigate to the location of the .v, .ve, or .inc files you want to check. Highlight the file names and click on the Add button. Then Click on the OK button to confirm your selections. You now see the full paths to the directories or files you specified displayed in the Directories or Files panes.
9. When you are done specifying your HDL source files, click on the Next button at the bottom of the window to proceed to the next step, Confirm & Create.
10. The Confirm and Create window has a Build and Check check box at the top that is selected by default. If you deselect this check box, Leda analyzes your HDL files and builds your project as specified in the Wizard, but does not run the Checker. Leave this check box alone if you want Leda to also run the Checker after building your project. Either way, click the Finish button at the bottom of the window to proceed. If the tool displays a small Get Top Module/Design/Entity window, note that this information is needed for checking chip-level rules. Specify the top module or design entity and click on the OK button. Leda compiles the source files and

executes the Checker. You should see something like the following screen (Figure 18). This example shows the results for a mixed-language project that contains both Verilog and VHDL source files.

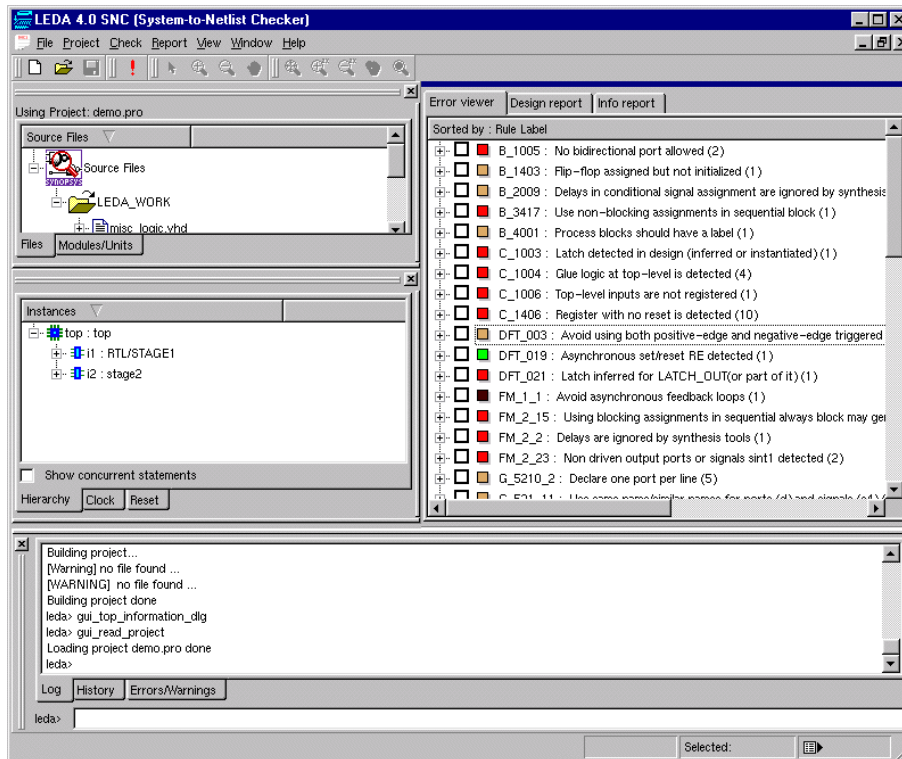


Figure 18: Leda Checker Results

Propagating Constants

If your design supports test mode, you can reduce the number of false errors reported by the Leda Checker by specifying constants for primary inputs used in test mode (see [Figure 19](#)).

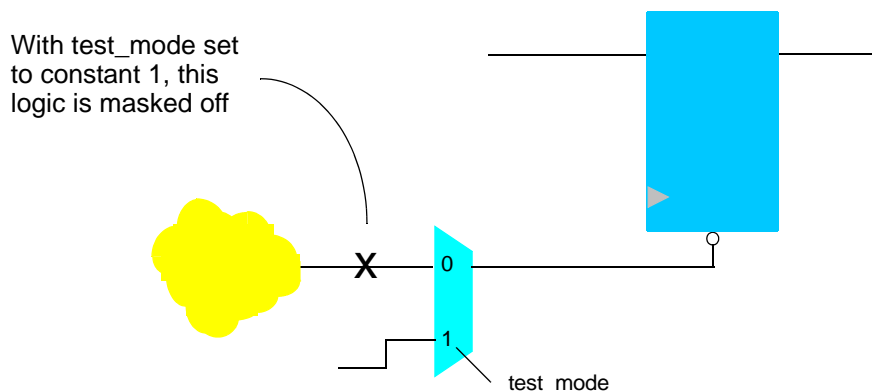


Figure 19: Constant Propagation for Test Mode

For the circuit shown in [Figure 19](#), Leda flags the gated reset as an error. However, if you set the `test_mode` signal to a constant 1, both the mux and the “false” input are masked off from the analysis, eliminating the false error.

Constant propagation is therefore particularly useful for design for test (DFT) rules, for which Leda provides a prepackaged policy (see the *Leda Prepackaged Rules Guides*).

You can set constants for primary inputs and internal signals using the `set_case_analysis` command in a simple ASCII text file. The syntax is:

```
set_case_analysis <value> <pin_or_port>
set_case_analysis <value> {<list of pin_or_port>}
```

Where:

```
<value> = 0 | 1 | zero | one
<pin_or_port> = /instantiation/hierarchy/to/internal/<pin> | <port>
```

For example:

```
set_case_analysis 0 P1
set_case_analysis 0 U1/P2
set_case_analysis 0 {P1, U1/P2}
set_case_analysis 0 P1(0)
```


In accordance with Design Compiler's syntax, when you assign constants, make sure top-level signals do not have hierarchical names. For example:

```

module top();
    wire en1, clk, rst, in1;
    reg out1;
    inst I (en1, clk, rst, in1, out1);
endmodule

module inst (en1, clk, rst, in1, out1);
    input en1, clk, rst, in1;
    output out1;
    reg out1;

    wire en2, in2;
    assign gated_clk=clk & in2;
    assign gated_rst=rst & in2;
    assign int_clk=(en1)?clk:gated_clk;
    assign int_rst=(en2)?rst:gated_rst;
    always @(posedge int_clk or posedge int_rst) if (int_rst) out1<=1'b0;
    else out1<=in1;
endmodule

```

For this example, you set constants for the en1 and in2 signals as follows:

```

set_case_analysis 1 en1          <== No hierarchical info allowed here
set_case_analysis 0 I/en2

```

The signals you set with the set_case_analysis command take the specified fixed values for subsequent Leda Checker runs.



Note

When you propagate constants, Leda also propagates the values for supply0 and supply1 signals throughout your design. If there is a conflict, values you specify in your constraint file with the set_case_analysis command take priority.

If you are using the Leda Checker in batch mode, use the -case_analysis_file option to point to the ASCII text file that contains your set_case_analysis commands.

If you are using the Leda GUI, set constants for Leda to propagate using the Specify Design Information window that comes up when you invoke the Checker (see [Figure 24 on page 109](#)). Specify the file that contains your set_case_analysis file commands.

If you are using Leda in Tcl shell mode, use the set_case_analysis command to propagate constants (see [“set_case_analysis” on page 304](#)).

Constant Propagation Limitations

Although Leda accepts subelements in the constraint file, it currently does not propagate constants for them. The current version also does not propagate values for buses or internal constants. Use simple scalar values to specify constants for primary input signals only. Also, note that syntax errors in the constraint file currently cause the tool to crash.

Using the Rule Wizard to Select or Deselect Rules

With a project built, you can customize the Checker to run just the rules that you are interested in against your VHDL and Verilog source code. First, set up your environment for running the Checker (see “[Leda Environment Variables](#)” on page 317). Next, define any macros needed for expanding rules prior to checking, as explained in “[Defining Macro Values for Rules](#)” on page 82. Then from the main menu, choose **Check > Configure**. This opens the Rule Wizard (see [Figure 20](#)).

Configuration

Loaded

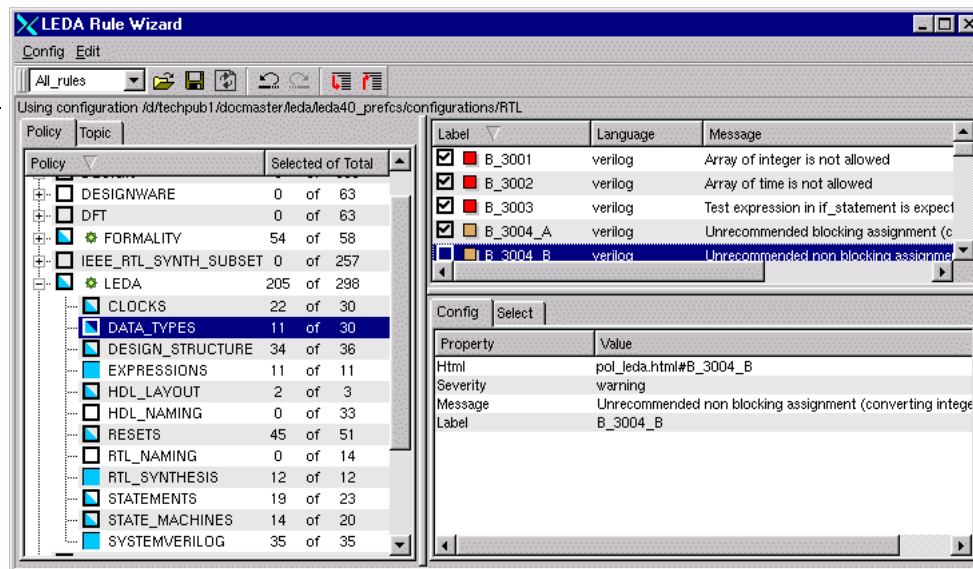


Figure 20: Rule Wizard Window



Note

If you write custom rules using the Tcl or C APIs, be sure to specify error message text in the VerSL rule wrappers. Otherwise, the rules are not visible in the Rule Wizard.

Using Prebuilt Configurations

To load a different rule configuration, from the Rule Wizard choose **Config > Load configuration**, and use the pull-down menu to select one of the prebuilt configurations:

- Gate-level—contains 90 chip-level and netlist/design rules selected from the Design and Leda general coding guidelines policies.
- Leda-classic—contains roughly 1,300 rules drawn from every prepackaged policy except DesignWare and STARC. It is close to the default configuration used in previous versions of the tool.
- Leda-optimized—contains roughly 1,100 rules from the same policies as Leda-classic. This configuration is “optimized” to remove similar rules from different policies.
- RTL—contains about 70 rules drawn from the DC, DFT, Formality, RMM, and Leda general coding guidelines policies. This configuration is the default.
- Custom—if you choose Custom, use the Load configuration window that pops up to navigate to and select your custom configuration.
- sdc-rtl—contains rules for SDC checks at the RTL level. A duplicate of this configuration is also available as sdc-quality-rtl prebuilt configuration.
- sdc-prelayout—contains rules for SDC checks at the prelayout level. A duplicate of this configuration is also available as sdc-quality-prelayout prebuilt configuration.
- sdc-postlayout—contains rules for SDC checks at the postlayout level. A duplicate of this configuration is also available as sdc-quality-postlayout prebuilt configuration.
- sdc-top_versus_block—contains rules to verify consistency between block level and top level design constraint file.
- sdc-equivalency—contains rules to verify equivalency between two SDC files of a design.

You can also load the default configuration (same as RTL prebuilt configuration) from the Rule Wizard by choosing **Config > Load Default**.

For information on using the SDC checker, see [“Using the SDC Checker” on page 133](#).

For lists of all rules contained in the major prebuilt configurations, see [“Leda Prebuilt Configurations” on page 321](#).

Policy and Topic Views

The Rule Wizard has several tabs and panels. The Topic tab on the left side lists rule topics in general categories that span multiple policies (for example, Clocks). The Policy tab shows you the policies that contain these rules. The two tabs provide different views of the same database of prepackaged rules. The top panel on the right side is blank until you either select a ruleset from within a policy in the Policy tab or from the general categories in the Topic tab. Then the top panel on the right fills up with all the rules from the selected ruleset. Click on the Label, Language, and Message bars to sort the display on any of these items in ascending or descending order. To deselect a rule for checking, click on the check box. When you click on another rule, the check box appears blank, confirming that the rule is now deselected for checking. To select a rule for checking, click on the blank check box. When you click on another rule, the check box appears with a check mark inside, confirming that the rule is now selected for checking.

Selecting or Deselecting Rules

Use the Policy and Topic tabs on the left side of the window to select or deselect entire policies, topics, or rulesets from either the policy or topic point of view. The Policy and Topic displays give you a hierarchical way to navigate through the available rules. Use the (+) or (-) box icons on the left side of each tab to expand or collapse the hierarchical display. You can also use the pull-down menu on the toolbar to change the view to All Rules, Verilog Rules, or VHDL Rules. Notice that when you change the selection status for a ruleset on the left side of the window, how your change is reflected in the individual rule display on the right side of the window.

The box icons in the Policy tab on the left side of the Rule Wizard display tell you the rule selection status for the rules in each policy:

- An open box (clear) indicates that all rules in that policy are deselected for checking
- A full box (set) indicates that all rules in that policy are selected for checking.
- A half-full box (partially set) indicates that some, but not all rules in that policy are selected for checking.
- A green star ✨ next to the policy selection box indicates that the policy has a subset of recommended rules selected for checking. The green star only appears when you deselect a policy in the Rule Wizard window and then select it again for checking. If you use the Wizard to change any of the defaults for that policy, the green star goes away.

You also select and deselect rules using in Tcl shell mode. See the command reference information for [“rule_deselect” on page 216](#) and [“rule_select” on page 239](#).

Disabling Redundant Rules

The Topics tab on the left side of Rule Wizard gives you an easy way to review all of the rules in the different policies related to a given topic. For example, if you click on the (+) icon next to the Clocks topic on the left side of the display, the tree expands to show a list of rules associated with clocks. One such rule is to avoid the use of both positive- and negative-edge triggered flip-flops in the same design. Because this is a good common sense design rule, it appears in several different policies. When you click on the (+) icon just to the left of the description for this rule, the display expands to show the different policies where this rule is available, including DFT, RMM, and STARC.

Let's say that you want a rule to be enabled for checking, but you don't want five different error messages to appear just because the rule is duplicated in five different policies. To narrow your error report display, you can use the Topics tab to view redundant rules. Click on the rule you want to disable and deselect it for checking using the check box on the right hand side of the Rule Wizard. Do this for all but one of the redundant rules, leaving just one relevant rule enabled for checking. To disable all redundant rules at once, first run the Checker, right click on the rule in the Error Viewer, and select "Disable Redundant Rules" from the pop-up menu.

Deactivating Rules

In addition to the Rule Wizard that you typically use before you run the Checker, Leda provides several other ways to deactivate rules: from a configuration file, directly in sections of HDL source code where you want rule checking temporarily turned off for specific rules, and even after a Checker run right from the Error Viewer. You can also use a command-line option to translate .leda_select file commands into Tcl commands that do the same thing. See the following sections:

- ["Deactivating Rules with a Rule Configuration File" on page 102](#)
- ["Deactivating Rules from within HDL Source Files" on page 103](#)
- ["Deactivating Verilint Policy Rules" on page 104](#)
- ["Deactivating Rules from the Error Viewer" on page 105](#)
- ["Deactivating Rules By File" on page 106](#)
- ["Translating .leda_select Files" on page 106](#)

Deactivating Rules with a Rule Configuration File

You can edit a rule configuration file (for example, `config.tcl`) to select or deselect policies, rulesets or rules. Point to this configuration file using the `-config path_to_file` batch option or the `$LEDA_CONFIG` environment variable.

To deselect a rule, use the following Tcl command in your configuration file:

```
rule_deselect -rule rule_label
```

For example, to deselect a rule labeled `B_1000`:

```
rule_deselect -rule B_1000
```

To deselect a rule only in certain HDL files, use the following Tcl command in your configuration file:

```
rule_deselect -rule rule_label -file file_name
```

For example, to deselect rule `B_1000` only in `myfile.v`:

```
rule_deselect -rule B_1000 -file myfile.v
```

The `rule_select` and `rule_deselect` commands in your configuration file are order-dependent. In the following example, the first command deselects all rules in the Leda policy, and the second command selects one individual rule (`B_1000`) from the Leda policy.

```
rule_deselect -policy Leda  
rule_select -rule B_1000
```

After reading these commands in your configuration file (or entered at the `leda> Tcl` mode prompt), the only rule that is selected for checking in the Leda policy is `B_1000`.

In this next example, the first command selects one rule in the Leda policy, and the second command deselects all rules in the Leda policy:

```
rule_select -rule B_1000  
rule_deselect -policy Leda
```

After reading these commands in your configuration file (or entered at the `leda> Tcl` mode prompt), all rules in the Leda policy are deselected for checking.

In this example, the first command deselects an individual rule in the Leda policy and the second command selects all rules in the Leda policy:

```
rule_deselect -rule B_1000  
rule_select -policy Leda
```

After reading these commands in your configuration file (or entered at the `leda> Tcl` mode prompt), all rules in the Leda policy are selected for checking, because the second command overrode the first one.

To turn off all rules in all policies in your configuration for checking, use the `-all` switch:

```
rule_deselect -all
```

To turn all rules back on for checking:

```
rule_select -all
```

In cases where two or more lines provide conflicting information, the last line read in the configuration file takes precedence. For more information on Tcl commands that you can use to manage rule configurations in Leda, see [“Rule Tcl Command Reference” on page 197](#).

Deactivating Rules from within HDL Source Files

You can deactivate rule checking for certain blocks of code using a configuration file to specify the file names and numbers concerned (see [“Fixing Errors Found by the Checker” on page 112](#)). However, this means that every time you modify that source file, you also have to update the configuration file in your configuration directory.

Another approach to deactivating all rule checking for selected blocks of code is to add the `“leda off”` and `“leda on”` pragmas to your HDL code. For example:

```
Module m (a, b, c):  
  Input a, b;  
  // leda off  
  Output c:  
  // leda on
```

means that the line `“Output c:”` does not get checked. Be sure to keep the `“leda off”` and `“leda on”` pragmas paired up around blocks of code that you want to turn off for checking. If you forget to include a `“leda on”` pragma at the end of a block of code turned off for checking the rule is disabled until the end of that file.



Note

`“leda off”` and `“leda on”` pragmas can be used to disable block-level, chip-level and netlist rules on a sub-part of the design.

You can also use these pragmas in a similar way to disable one or more individual rules for selected sections of your HDL code. For example, if you enter the following pragmas before a section of code in which you want to temporarily turn off rule checking:

```
// leda rule_1 off  
// leda rule_2 off  
// leda rule_3 off
```

then Leda does not report on errors for those rules you specified, where `rule_1`, `rule_2`, and `rule_3`, are valid rule labels.

If you then insert a “leda on” pragma for *rule_1* later in your source code, as follows:

```
// leda rule_1 on
```

only *rule_2* and *rule_3* are disabled for checking from that point forward. For each “leda off” pragma that you insert in your code to turn checking off for a specified rule, be sure to enter a matching “leda on” pragma further on in the file, as shown below for our example:

```
// leda rule_2 on
// leda rule_3 on
```



Note

The leda off/on pragmas can be disabled using the `-ignore_rule_pragmas` option in the command line.

Deactivating Verilint Policy Rules

If you want to deactivate rules just from the Verilint policy, use the following pragmas in your source code:

```
Module m (a, b, c):
  Input a, b;
  // verilint off
  Output c:
  // verilint on
```

The verilint off | on pragmas work just like leda off | on. The only difference is that they apply only to prepackaged rules in the Verilint policy. You can use these pragmas to disable one or more individual Verilint rules for selected sections of your HDL code. For example, if you enter the following pragmas before a section of code in which you want to temporarily turn off Verilint rule checking:

```
// verilint rule_1 off
// verilint rule_2 off
// verilint rule_3 off
```

then Leda does not report on errors for those rules you specified, where *rule_1*, *rule_2*, and *rule_3* are valid rule labels without the letter prefixes (use 410 for rule label W410).

If you then insert a “verilint on” pragma for *rule_1* later in your source code, as follows:

```
// verilint rule_1 on
```


only *rule_2* and *rule_3* are disabled for checking from that point forward. For each “verilint off” pragma that you insert in your code to turn checking off for a specified Verilint rule, be sure to enter a matching “verilint on” pragma further on in the file, as follows:

```
// verilint rule_2 on
// verilint rule_3 on
```

For example, if the label of a Verilint rule you want to temporarily turn off for checking is W410, specify just the number with the pragma:

```
// verilint 410 off
...
// verilint 410 on
```

For more information on the Verilint policy, including the individual rule labels, see the [Leda Verilint Rules Guide](#).

Deactivating Rules from the Error Viewer

Another way to deactivate checks for individual rules or all identical rules is to use the Error Viewer after you run a check. Right click on the error message that you want to deactivate, and select “Disable the rule” or “Disable redundant rules“. This causes Leda to dim the affected messages and deactivate those rules for the next Checker run, as shown in [Figure 21](#).

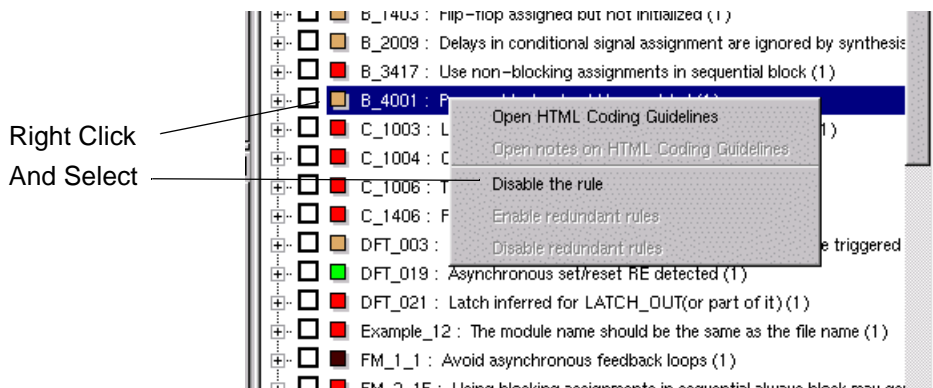


Figure 21: Deactivating Rules from Error Viewer

Deactivating Rules By File

You can use the Select tab in the bottom right panel of the Rule Wizard to deactivate rules only for specific HDL source files (see [Figure 22](#)).

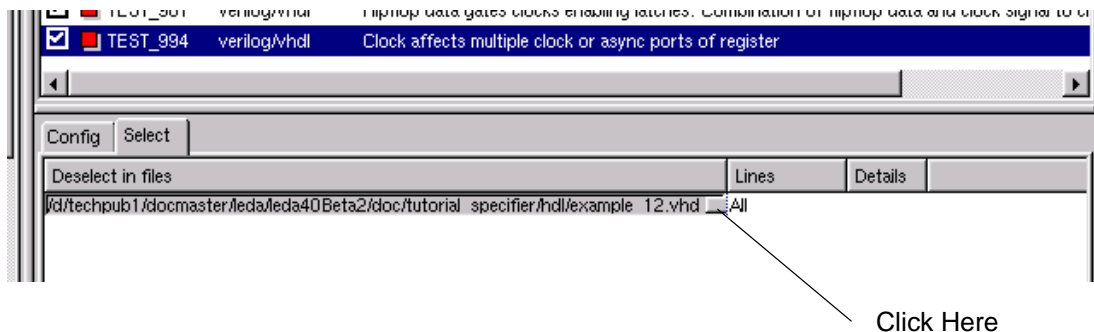


Figure 22: Deselect Rules by File in Rule Wizard

First, select the rule that you want to deactivate in the top-right panel of the Rule Wizard. Then click the small rectangular button in the Select tab. Use the Deactivate in File window that comes up to navigate to and select the file where you want that rule deactivated for checking. By default, the rule is now deactivated for the entire file, as shown by the “All” in the Lines column. To deactivate the rule only for specific lines in the file, click on the “All” string. This activates a pop-up menu where you can select “Lines” instead.

Translating .leda_select Files

Leda no longer supports .leda_select files, but you can use a built-in translator to read .leda_select files and translate them into files that contain equivalent Tcl commands. To invoke the translator, run Leda with the -upgrade400 switch:

```
% leda -upgrade400 [-select file]
```

When you use this command, Leda looks for .leda_select files in the following locations:

- \$LEDA_SELECT_FILE
- \$cwd
- \$HOME
- \$LEDA_PATH/.leda_config

Leda puts the output .tcl file in a .leda_select.tcl directory in the same directory where it found the .leda_select file. Or, if you have \$LEDA_SELECT_FILE set or use the old -select *file* option on the command line when you invoke the translator, Leda creates a *file.tcl* file in the specified directory.

In all cases, you must copy-and-paste the new Tcl commands from the output file created by the translator into a configuration file in your configuration directory (see [“Deactivating Rules with a Rule Configuration File” on page 102](#)).

During normal execution, Leda issues a warning message if it finds a .leda_select file in any of the following locations:

- -select *file* (on the command line)
- \$LEDA_SELECT_FILE
- \$cwd
- \$HOME
- \$LEDA_PATH/.leda_config

Setting & Saving Checker Preferences

When you finish selecting the rules you want to check, set your Checker preferences.

1. From the main menu, choose **File > Preferences**. This brings up the Application Preferences window. From the list on the left, choose Checker (see [Figure 23](#)).

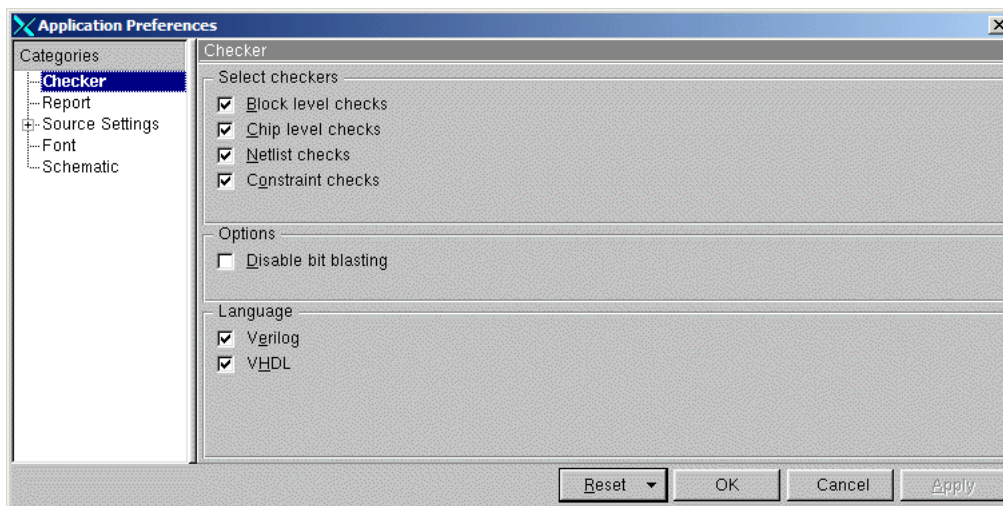


Figure 23: Checker Options in Application Preferences

2. All checks are enabled by default (block-level, chip-level, netlist, and constraint). To disable any of these classes of checks, deselect the associated check boxes.
3. If you do not want Leda to treat signals in vectors or buses individually, select the Disable bit blasting check box. This might be useful if you only want to run block-level checks and want to speed up the performance of the tool.




Caution

Don't disable bit blasting if you are running netlist checks. They may not work right with bit blasting disabled.

4. When you are done setting your Checker preferences, click the OK button.
5. To set your application preferences back to the defaults, click the Reset button.

To save your Checker preferences, choose **File > Save Preferences**. Leda saves your preferences in a `$HOME/.synopsys_leda_prefs.tcl` file. The next time you invoke the GUI, Leda uses these preference settings if the file exists. Otherwise, Leda uses the default preferences. You can also configure Leda to automatically save the Checker preferences you specify using the **File > AutoSave Preferences** toggle switch.

Running the Checker

From the main menu, choose **Check > Run** or click the run icon  on the toolbar. This brings up the Specify Design Information window (see [Figure 24](#)), which has three tabs ([Top Unit Tab](#), [Test Clock/Reset Tab](#), and [Checkers Tab](#)). Work the tabs from left-to-right.

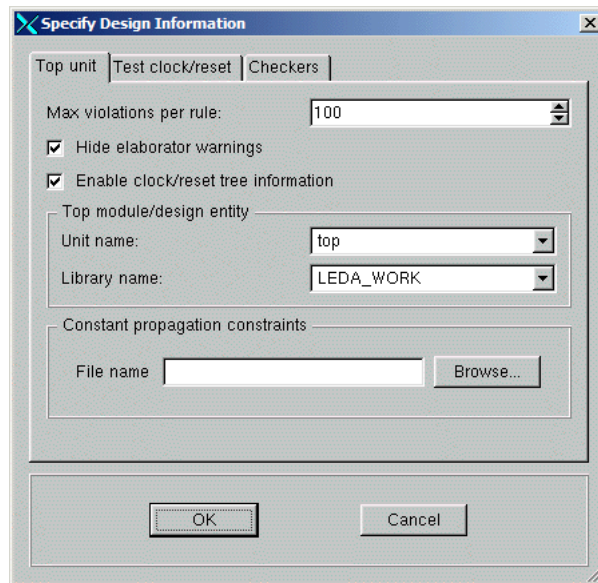


Figure 24: Specify Design Information Window (Top Units Tab)

Top Unit Tab

1. Select the Top unit tab and specify the “top” file in your design using the Unit name and Library name pulldown menus.
2. If you want Leda to display warning messages from the elaborator when performing chip-level checks, deselect the Hide elaborator warnings check box at the top left of the Top unit tab. Otherwise, Leda’s default behavior is to hide warning messages from the elaborator.
3. The Max violations per rule is set to 100 by default. If you want Leda to show a larger or smaller number of violations per rule, use the dial-up/down menu to set the new number.
4. If you want to propagate constants in the design for this Checker run, enter the full path to your constant propagation file in the Constant propagation constraints panel, or use the Browse button to navigate to and select your constant propagation file (see [“Propagating Constants” on page 96](#)).

5. If you are running Synopsys Design Constraint (SDC) checks, use the Synopsys Design Constraints panel to enter the full path to the SDC file that you want to check. For information about using the SDC checker, see [“Using the SDC Checker” on page 133](#).

Test Clock/Reset Tab

1. To create test clocks or resets from ports named in the Specify Design Information window, click the Test clock/reset tab. This changes the display as shown in [Figure 25](#). Note that generation of data for the Clock and Reset Tree browsers can slow Leda’s performance on large netlists. Use this feature only when needed. It is off by default.

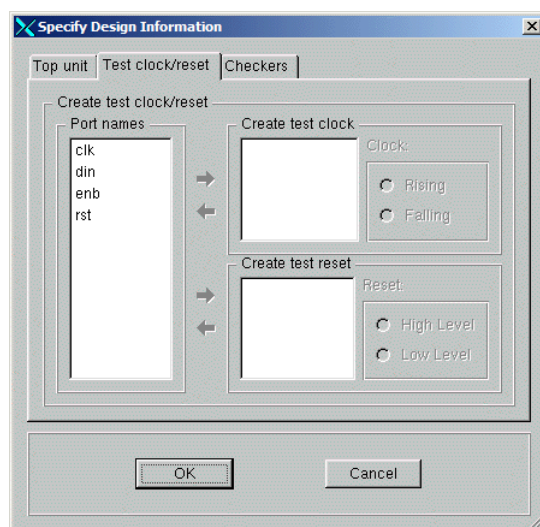


Figure 25: Test Clock/Reset Tab

2. For each test clock that you want to create:
 - m Select a port name from the Port names listbox.
 - m Click on the arrow button to add the selected name to the Create test clock listbox.
 - m Specify Rising or Falling edge for the clock cycle using the radio buttons in the Create test clock panel.
3. For each test reset that you want to create:
 - m Select a port name from the Port names listbox.
 - m Click on the arrow button to add the selected name to the Create test reset listbox.
 - m Specify the first level (High or Low) for the scan shift phase.

Checkers Tab

The initial values displayed in this tab are inherited from the Application Preferences window Checker settings (see [Figure 23](#)).

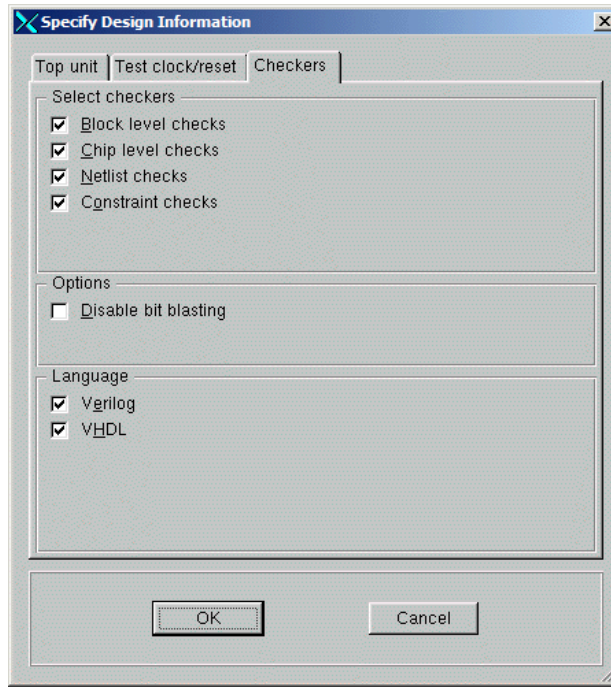


Figure 26: Checkers Tab

Changes that you make here in the Checkers tab only affect the current run with the tool.

1. All checks are enabled by default (block-level, chip-level, netlist, and constraint). To disable any of these classes of checks, deselect the associated check boxes.
2. If you do not want Leda to treat signals in vectors or buses individually, select the Disable bit blasting check box. This might be useful if you only need to run block-level checks and want to speed up the performance of the tool.



Caution

Don't disable bit blasting if you are running netlist checks. They may not work right with bit blasting disabled.

3. When you are done making all your selections, click the OK button. The Checker compares your HDL source files against the policies or rules you selected, and displays the results.

In GUI mode, Leda saves the Tcl commands from your setup and configuration files in a `leda_history.log` file in the current working directory. This file is overwritten for each new session.

For more information on Chip-level checkers and Netlist checkers, see [Table 3](#)

Fixing Errors Found by the Checker

After you run the Checker on a design, the main window fills up with a lot of detailed information about your HDL design, including a complete hierarchy of all your source files, a summary panel that reports the number and kind of errors found, and an Error Viewer that you can use to learn more about the rules that were violated using the HTML-based help system. There is also a Path Viewer and Clock and Reset Tree browser that help you visualize errors for rules where tracing information is available. Best of all, you can also hyperlink directly from the Error Viewer to the exact locations in your HDL code where Leda spotted rule violations, make the fixes, and recompile your project without leaving the tool. [Figure 27](#) shows the Checker window after running the demo project that comes with Leda.

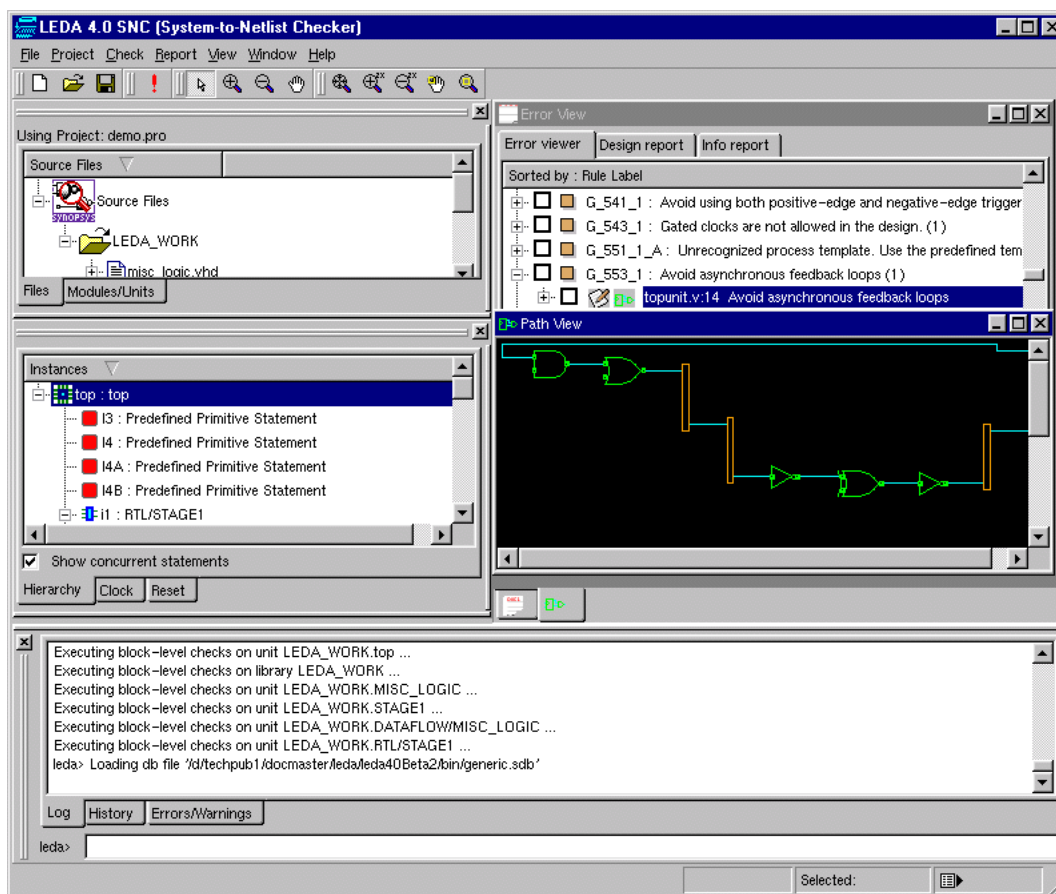



Figure 27: Checker After Check

You can use the Checker's Error Viewer to review and fix errors found using the Checker GUI mode (described here) or the command-line Checker (see [“Using Leda Batch Mode” on page 145](#)). To fix your HDL code as Leda suggested, follow these steps:


1. For each warning or message in the error report, click on the (+) box icon to the left of the message in the Error Viewer. This expands the display to show the HDL source file for the rule that was violated. When you click on the (+) box icon at this next level, Leda displays part of the HDL file that was tested.

The offending line of code is indicated with a green pointer  .

2. If tracing information is available for a violated rule, a logic gate icon appears next to the file name and error message. To make the Path Viewer appear in the lower half of the Error Viewer, click on the green logic gate icon  . Use this view of the error to visualize the circuit path causing the problem.

Note that when you have both the Path Viewer and Error Viewer windows open, you can click on the green logic gate icon for any other chip-level rule in the Error Viewer, and the Path Viewer changes to stay in sync with the rule you are debugging.

3. For each warning or error message in the Error Viewer, double-click on the line of code next to the green pointer. This opens a text editor on the file. The suspect code is already highlighted in the file. Make your fixes and then choose **File > Save** from the editor's window to save your changes.

You can also use the  icon next to the error message to split the top of the window into a view of your HDL source file on the right, and the error messages on the left. Note that you can't edit the HDL source file from this view.

4. For each warning or error message in the Error Viewer, use the check boxes in the display to keep track of the violations that you have debugged and fixed (see [Figure 27](#)). Left click in a check box to make a check mark appear. Left click again to toggle back and clear the check box. When you later save the log file, the status of the check boxes for each violation is also saved in the log file. This way, when you reload the log file (leda.log) the check boxes that you marked and saved from your last session with the tool are displayed.
5. When you are done fixing the errors that you consider to be significant, choose **Checker > Run** again from the main menu. Leda recompiles your HDL files and checks them against the rules that you have activated. This time, since you corrected the troublesome HDL code, your results come up clean, with no messages listed in the Error Viewer.

Reviewing Log, History, Errors/Warnings Tab in GUI

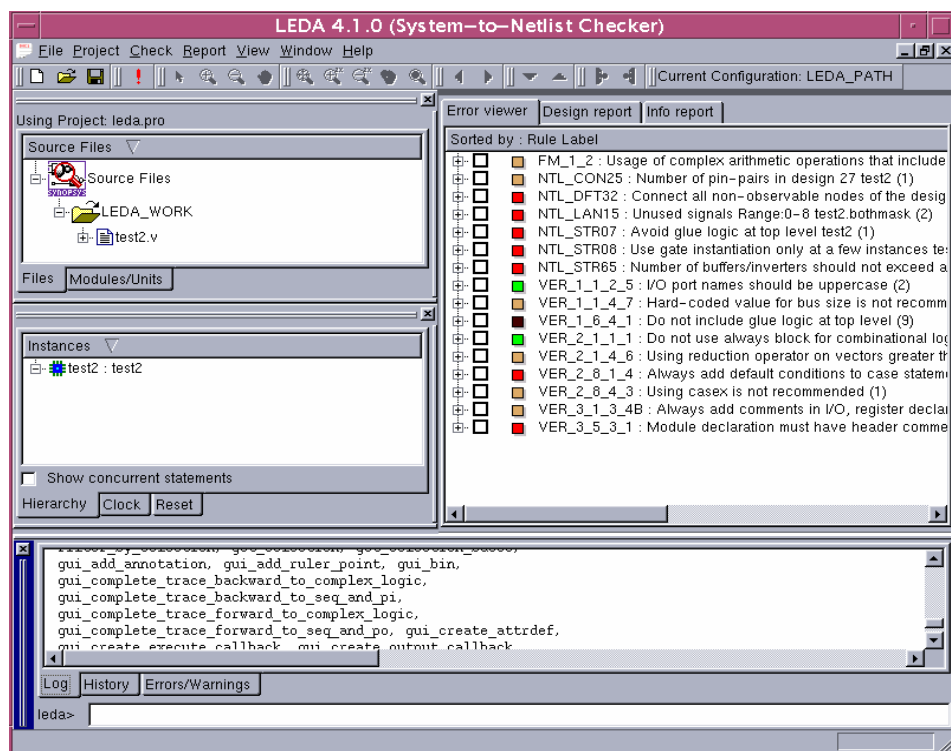


Figure 28: Log, History, Error/Warnings Tab

The Log tab present in the Leda GUI displays the processing messages and the results of commands executed.

The History tab present in the Leda GUI displays the list of all commands that were executed during the current session in sequence. You can also see “Edit” tab and “Execute” tab present above the “History” tab. If you want to execute any of the commands used earlier, choose the command from the list and click “Execute”. Similarly, if you want to modify any of the previously used command, choose the command from the list and click “Edit”. The selected command appears in the Tcl command line. Now you can edit it and to execute it, press Enter key.

The Error/Warnings tab shows the errors/warnings encountered while executing a Tcl shell command given by a user. These errors/warnings are the result of unsuccessful/incomplete execution of a given command. For more information, see [“Invoking the Checker/Specifier GUI” on page 170](#)

Displaying Error Messages for STARC Policies

The messages displayed in the Error Viewer appear in English by default for all prepackaged policies. However, the VER_STARC_DSG and VHD_STARC_DSG policies contain a special feature that you can use to get error messages in Japanese for these policies. To configure the Error Viewer for Japanese error messages, set the LEDA_LANGUAGE environment variable to JAPANESE before you invoke the Checker tool, as shown in the following example:

```
% setenv LEDA_LANGUAGE JAPANESE
```



Note

This feature only works for the VER_STARC_DSG and VHD_STARC_DSG prepackaged policies. Also, the Japanese error messages do not appear in the Leda Rule Wizard.

Getting Prepackaged Rule Help for STARC Policies

Prepackaged rule help in HTML is available for the VER_STARC_DSG and VHD_STARC_DSG policies in both English and Japanese. To set the language for the HTML rule help to Japanese, follow these steps:

1. Navigate to the html directory in the Leda installation tree:

```
% cd $LEDA_PATH/doc/html
```

2. To set the HTML rule help to Japanese, create symbolic links as follows:

```
% ln -s jpn/dsg_ver_jpn dsg_ver
```

```
% ln -s jpn/dsg_vhd_jpn dsg_vhd
```

3. To set the HTML rule help back to English, create symbolic links as follows:

```
% ln -s eng/dsg_ver_eng dsg_ver
```

```
% ln -s eng/dsg_vhd_eng dsg_vhd
```

Note that English is the default configuration.

Sorting the Error Viewer Display

You can sort the results displayed in the Checker’s Error Viewer in a variety of ways, as discussed in this section.

You can configure your preferences for the display by choosing **File > Preferences** from the main window. This brings up the Application Preferences window (see [Figure 29](#)), which has several categories listed in the panel on the left-hand side of the display. Click on the **Report** category to view and edit your Error Viewer preferences. Specify how you want the display sorted (by Label, Master Rule, File, Policy, Module/Unit, Severity, or Master Rule). The default is Label. You can also sort the display using **Report > Sort by** from the main menu.

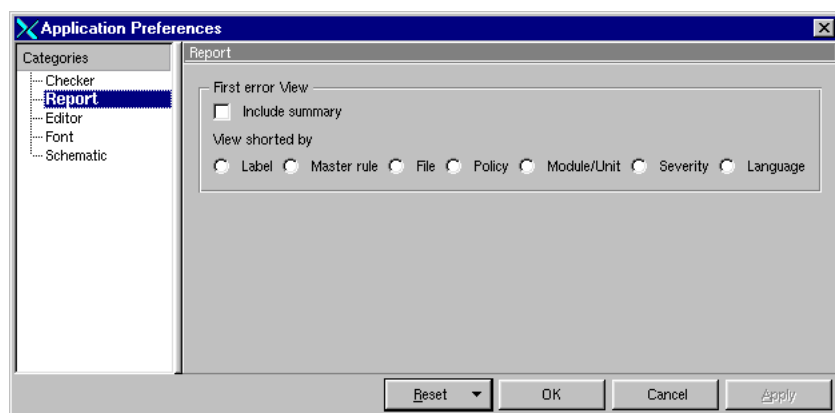


Figure 29: Error Viewer Preferences Window

To make Leda generate a summary report at the top of the Error Viewer, click on the “Include Summary” check box in the Report Preferences window. You can also toggle the display of the summary report using **Report > Summary** from the main menu. With the summary report open, you can click on any of the blue hyperlink totals to sort the display in the Error Viewer by that item (see [Figure 30](#)).

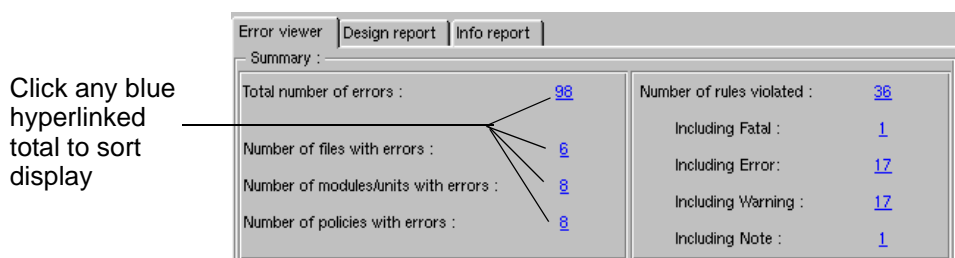


Figure 30: Error Viewer Summary

Filtering the Error Viewer Display

To filter the display in the Error Viewer, choose **Report > Filter by** from the main menu and select the frame of reference for your filter from the available options (Label, Master Rule, File, Policy, Module/Unit, Severity, or Master Rule). When you choose one of these options from the pulldown menu, Leda brings up an “Enter regular expression for filter” window. Enter a simple regular expression in the window, and click OK. For example, if you choose **Report > Filter by > Rule Label**, and then enter $\wedge B_$ in the regex window, Leda shows you just the violated rules that start with B_ in the Error Viewer.

Error Report Displays

The display you get in the Error Report depends on the sorts and filters that you have applied. Leda determines the most appropriate view based on your sorting and filtering selections. There are two basic types of displays:

- “Rule Display” on page 117
- “File Display” on page 119

Rule Display

In the rule display, the first level in the Error Viewer shows the severity, label, and message for each rule that was violated. Leda displays the total number of violations for each rule in parentheses at the end of this line. (see [Figure 31](#)).

Use check boxes to mark off errors as you debug them

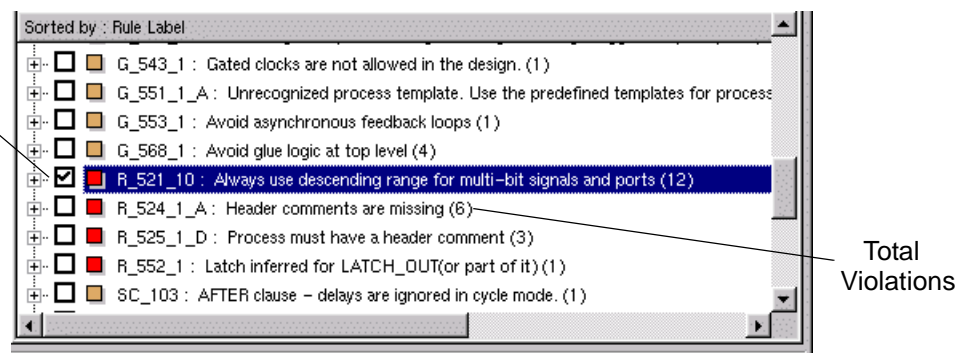


Figure 31: Severity, Message, and Label in Rule Display

The second level in the rule display shows all violations of the rule, listed by file name. From this view, you can hyperlink directly to a text editor and correct your source files (see [Figure 32](#)).

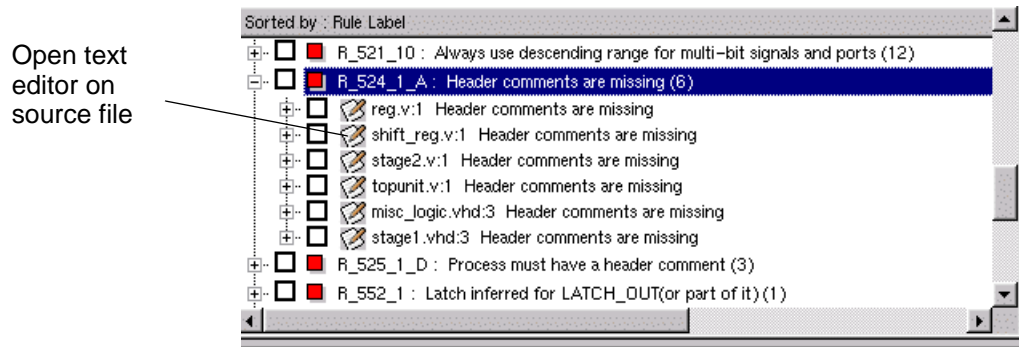


Figure 32: File Level in Rule Display

The third level in the rule display shows the HDL code where the rule violation was found, with the line indicated by a green triangle (see [Figure 33](#)).

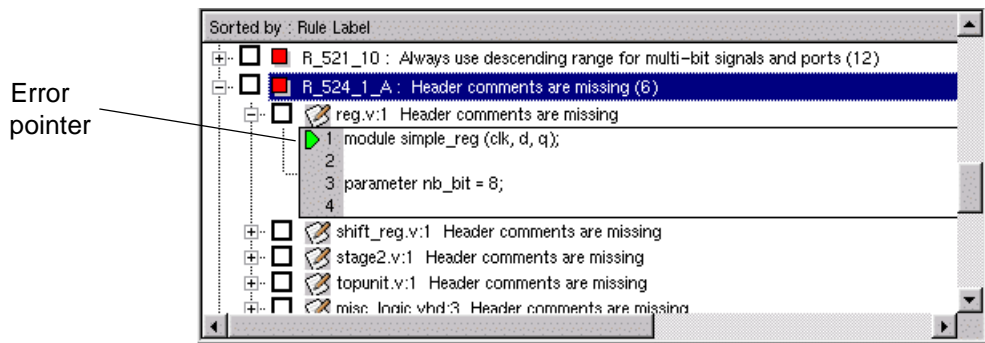


Figure 33: HDL Fragments in Rule Display

File Display

In the file display, the first level in the Error Viewer shows the file name and the number of violations found in that file in parentheses at the end of the line. The second level lists all the violations in that file. For each violation, the Error Viewer displays the severity, label, and message for the rule that was violated (see [Figure 34](#)).

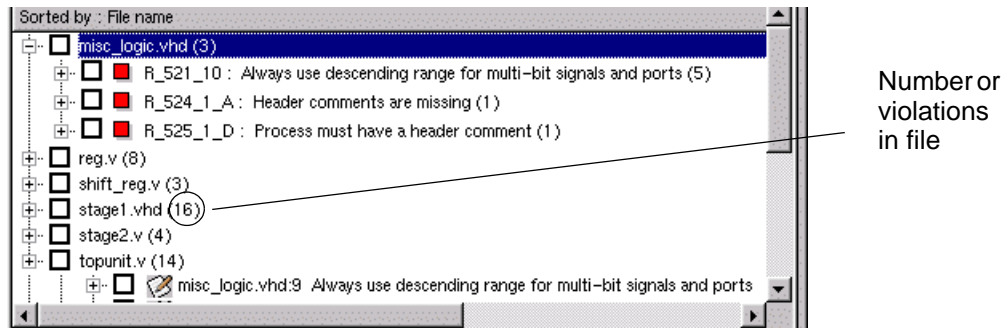


Figure 34: Error Level File Display

The third level shows the HDL code fragment where Leda found the violation (see [Figure 35](#)). The third level also shows tracing information for chip-level errors, if applicable. For information on chip-level tracing, see [“Using the Path Viewer”](#) on [page 121](#).

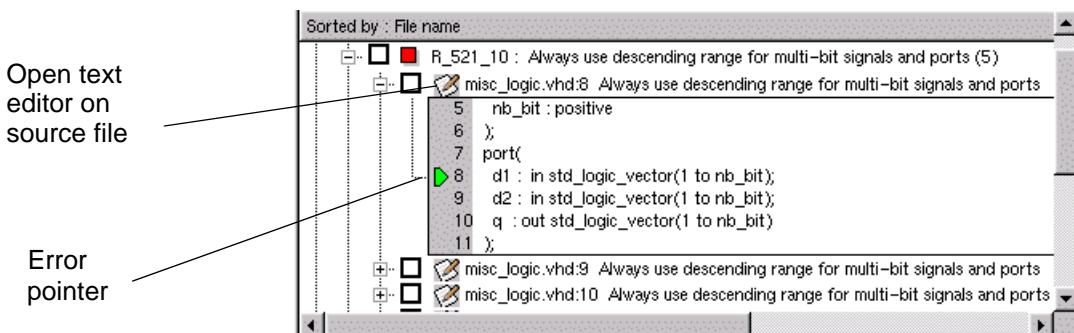


Figure 35: HDL Fragments in File Display

Viewing the Design Report

If you have any chip-level rules selected when you run the Checker, Leda produces a design report that provides detailed information about your design in the Design Report tab on the right side of the main window (see [Figure 36](#)).

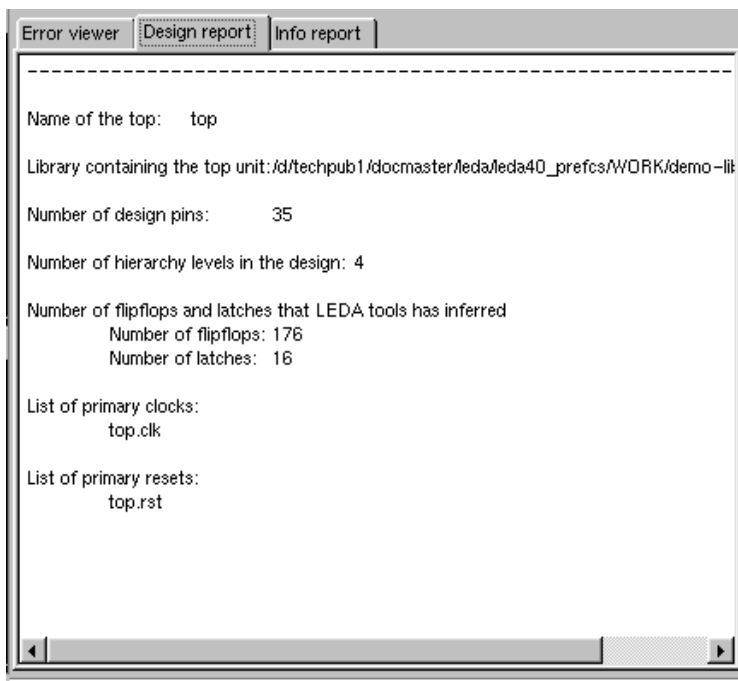


Figure 36: Leda Design Report

The Design Report tab provides information that includes:

- name of the top unit in the design
- number of pins in the top unit
- number of flip-flops and latches that Leda inferred in the design
- list of the primary clocks and resets

Using the Path Viewer

For some chip-level rules, it can be useful to visualize the sub-hierarchy that caused an error. For example, rules like the prohibition against asynchronous feedback loops may involve a connection that passes through several layers of hierarchy. Identifying the causes of such errors can be difficult with only source code and line numbers to help. To solve this problem, Leda provides a Path Viewer window that you can use to view connections over the entire design hierarchy, trace forward and backward in the schematic, and link directly from there to your source code.

When you violate a chip-level rule, the Error Viewer sometimes shows a small green circuit symbol (see [Figure 37](#)). To make the Path Viewer appear, click on the circuit symbol.

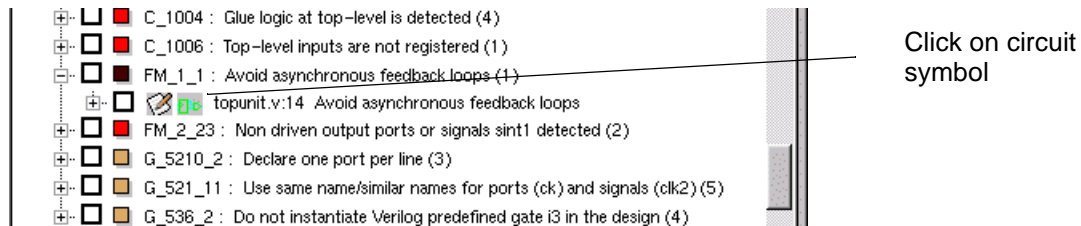


Figure 37: Invoking the Path Viewer

This brings up an integrated Path Viewer window in the lower half of the Error Viewer (see [Figure 38](#)).

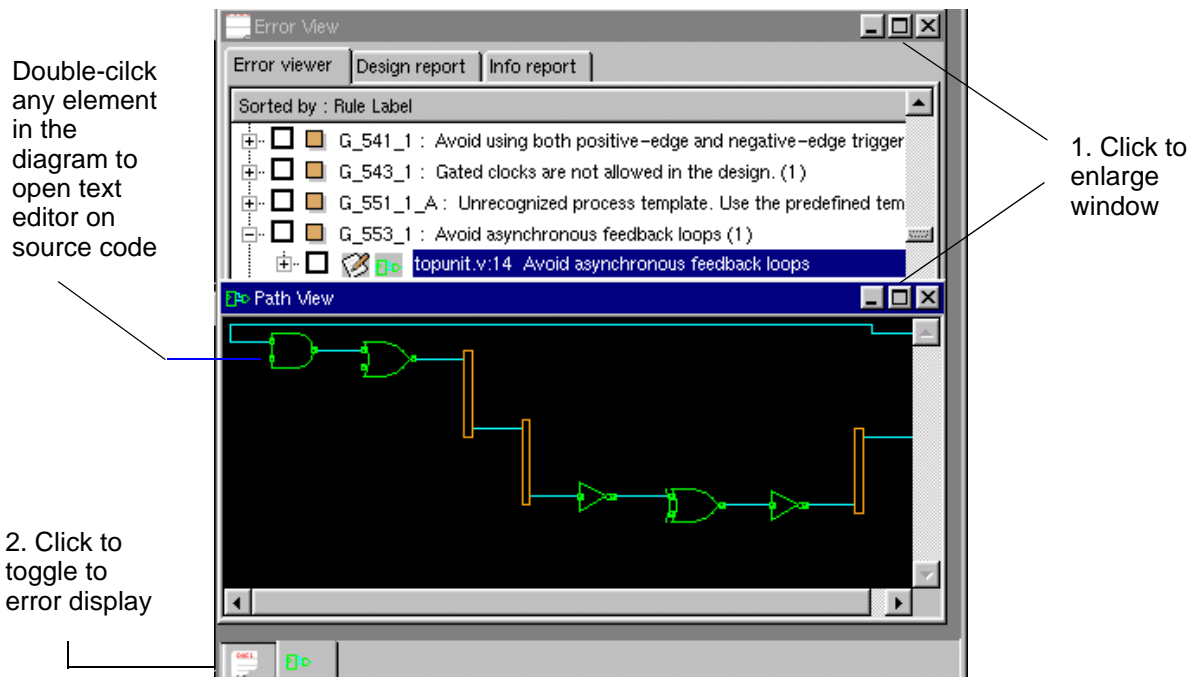


Figure 38: Path Viewer Window

If you cannot see all of the circuit diagram in the window, first enlarge the Path Viewer or Error Viewer by clicking on the maximize icon on the top right. You can then use the tabs at the bottom of the window to toggle back and forth between full views of the Error Viewer and Path Viewer.

To the left of the Path Viewer window there are three tabs (Hierarchy, Clock Tree, and Reset Tree). The Hierarchy tab on the far left shows each level of hierarchy in the design, including concurrent statements in each block. To toggle the display of concurrent statement information, select or deselect the check box next to the “Show concurrent statements” text in the lower left corner of the window (see [Figure 39](#)).

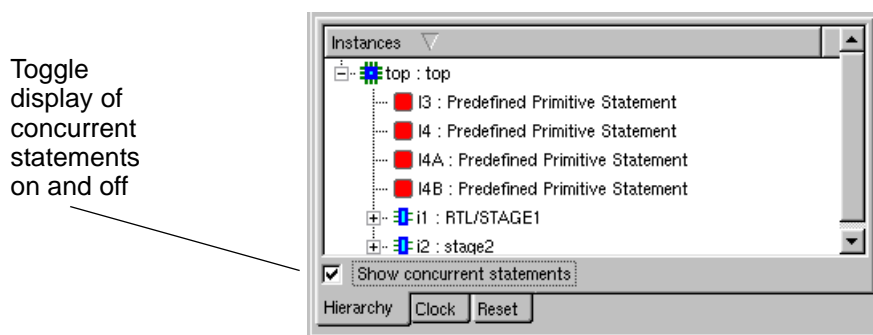


Figure 39: Hierarchy Browser Window

The colors on boxes in the hierarchy display have the following meanings:

- Red for concurrent statements.
- Blue for instantiations. (Double click on the (+) box icon next to any blue box to display the next level in the hierarchy.)
- Yellow for block and generate statements.

There are three types of hierarchy crossing symbols (see [Figure 40](#)).

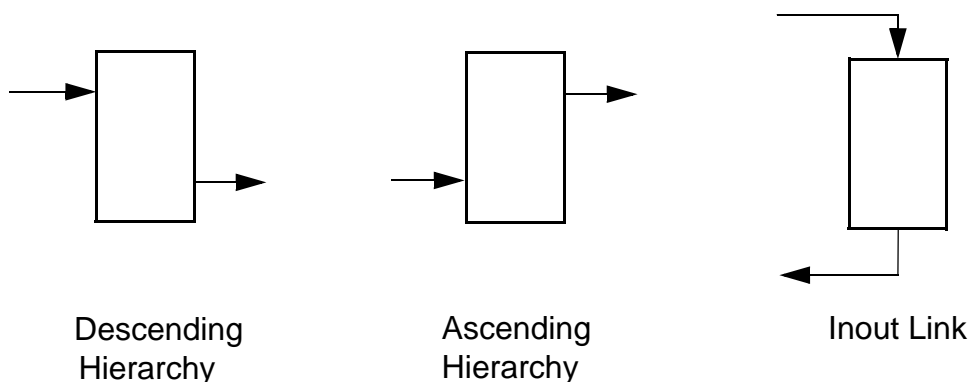


Figure 40: Hierarchy Types in Path Viewer

The Path Viewer on the right side of the display shows a circuit diagram of the connection that caused the violation, as well as the hardware that appears on this path. From this side of the window you can:

- Place the mouse pointer over any element in the diagram to display the full hierarchical name of the element.
- Single-click any element in the diagram to highlight the corresponding element in the hierarchical display on the left side of the window.
- Double-click any element to open the text editor on the corresponding source code in your design and modify as needed to correct the error.

The currently selected item in the Path Viewer displays in white.

Using Trace Forward and Trace Backward

You can trace forward and backward in the design using the Path Viewer starting from three kinds of objects (see [Figure 41](#)):

- **Pins.** Primary inputs can trace forward and primary outputs can trace backward. Cells and instance pins can trace forward and backward.
- **Cells.** All cells (built-in and .db) can trace forward and backward. You cannot trace instances.
- **Nets.** All nets can be traced. Note that some top-level instances may not show a result.

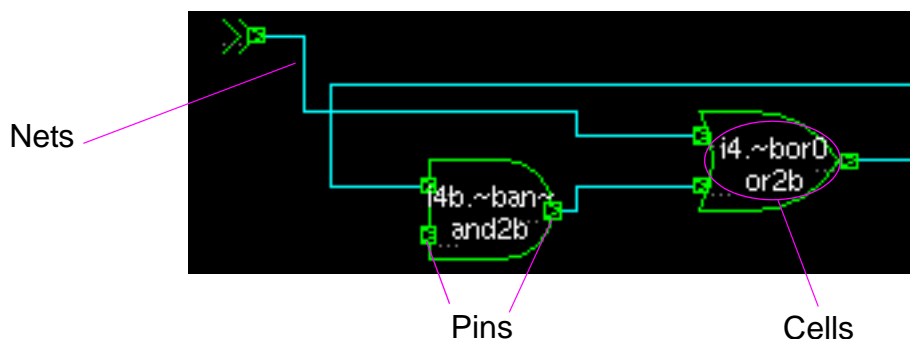




Figure 41: Traceable Objects in Path Viewer

Before you begin, make sure bit blasting is enabled (this is the default). To check this, choose **File > Preferences > Checker** and make sure the Disable Bit Blasting check box is not checked.

1. Left-click once on a graphical element in the Path Viewer window (for example, a net, flip-flop, gate, or pin of a flip-flop or gate). The currently selected graphical element turns white.
2. Notice that the trace forward  and trace backward  buttons on the GUI toolbar are now enabled. Click one of the buttons to trace forward or backward in the schematic. A new, larger Path Viewer window opens to show the results of the trace. You can move this standalone Path Viewer anywhere you want on your screen.

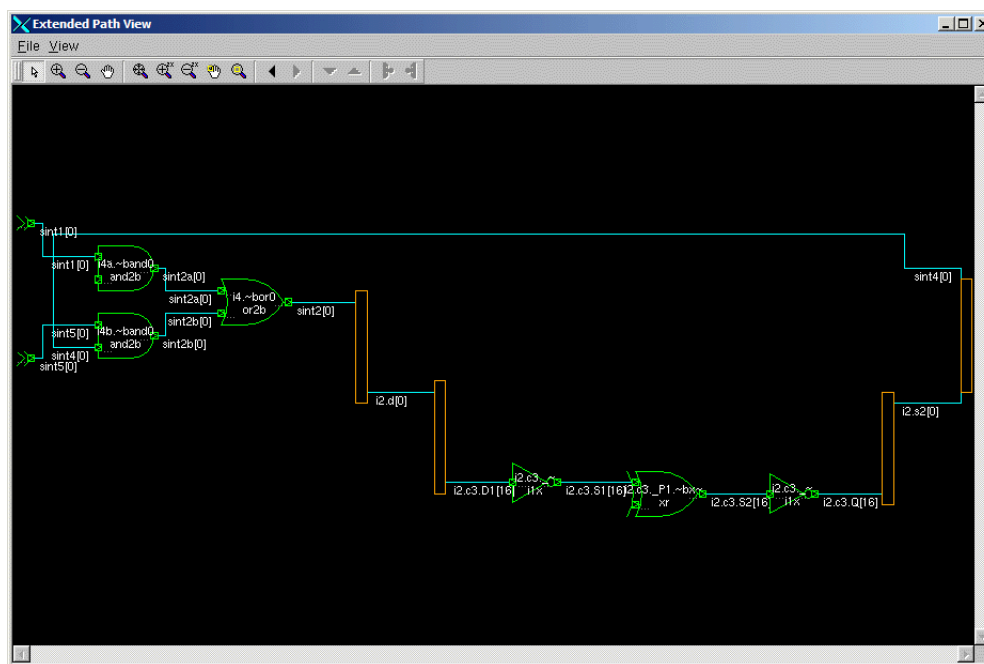










Figure 42: Extended/Standalone Path Viewer Window

If you prefer the schematic to display in the smaller Path Viewer window where you started the trace, choose **File > Preferences > Report**, and select the check box for “Use same path viewer for all operations.” Click the Apply button to make your change effective for the current session, and then click OK to dismiss the Application Preferences dialog box.

3. You can now trace forward  or backward  again in the standalone Path Viewer window after selecting an element from which to begin the next tracing operation.

**Note**

If you run Leda in batch mode and load the log file in the GUI to analyze the violations, the tool prompts you to see if you want to re-elaborate the design. This re-elaboration is required in order to use the tracing functions in the Path Viewer.

4. When the second trace displays, either the Previous Schema  or Next Schema  button on the Path Viewer toolbar is enabled, so that you can toggle back and forth between schemas. This is also useful if you want to examine traces for more than one violation at once. If you have multiple netlist or chip-level violations in the Error Viewer, click the green circuit symbol to bring up the Path Viewer again on another violation, left-click on a graphical element, and click the trace forward or trace backward button. The standalone Path View window changes to show the results of this most recent trace. You can now toggle back and forth between traces for the two different violations by clicking the Previous Schema and Next Schema buttons.
5. If you have a hierarchy crossing boundary symbol selected in the Path Viewer, (thin orange rectangle), you can use the move up  or move down  buttons on the toolbar to trace up or down one level in the hierarchy, depending on your location. Note that these buttons do not work after you use the Previous Schema  or Next Schema  buttons.

Scanning to Sequentials or Primary Ports (off by default)

If you want the trace to stop at the next sequential, primary port, or complex logic, choose **File > Preferences > Report**, and select the check box for “Scan to sequential and primary port.” Click the Apply button to make your change effective for the current session, and then click OK to dismiss the Application Preferences dialog box.

Extending the Current Schematic (on by default)

When you first use the trace forward or trace backward buttons, the standalone Path Viewer window comes up displaying the same circuit schema that you saw in the smaller Path Viewer window inside the GUI’s main window. But now you get an extended view of the schematic that also shows the result of the trace. From here, you can continue to trace forward or backward using the tool bar buttons. If you don’t want the first invocation of the standalone Path Viewer window to extend the current schematic, choose **File > Preferences > Report**, and deselect the check box for “Extend current schematic.” Click the Apply button to make your change effective for the current session, and then click OK to dismiss the Application Preferences dialog box.

Using the Clock and Reset Tree Browsers

To use the Clock and Reset Tree browsers, you must enable generation of the underlying data using the `-clockdump` and `-full_log` switches in batch mode or the Specify Design Information window (**Check > Run**) in GUI mode before running your checks. Then, when you click on the Clock or Reset Tree tabs in the Path Viewer, Leda displays the design hierarchies for clocks or resets in the left-hand side of the window (see [Figure 43](#)), and shows the signal paths through the area of your design where Leda flagged a rule violation in the schematic viewer on the right-hand side of the window.

Clock Tree	Flipflops	Latches
TOP : CLK	176	16
TOP : TOP	176	16
I1 : RTL/STAGE1	16	16
I3 : I3	160	0
CLK2 : CLK2		
I2 : STAGE2	160	0

The screenshot shows a window titled 'Clock Tree' with a tree view on the left and two columns on the right labeled 'Flipflops' and 'Latches'. The tree view shows a hierarchy starting with 'TOP : CLK', which expands to 'TOP : TOP', then 'I1 : RTL/STAGE1', 'I3 : I3', and 'CLK2 : CLK2'. Under 'I1 : RTL/STAGE1', there is a sub-entry 'I2 : STAGE2'. The 'Flipflops' and 'Latches' columns show counts for each level: 176 flipflops and 16 latches for 'TOP : CLK' and 'TOP : TOP'; 16 flipflops and 16 latches for 'I1 : RTL/STAGE1'; 160 flipflops and 0 latches for 'I3 : I3'; and 160 flipflops and 0 latches for 'I2 : STAGE2'. At the bottom of the window, there are three tabs: 'Hierarchy', 'Clock', and 'Reset', with 'Clock' currently selected.

Figure 43: Clock View in Clock and Reset Tree Browser

On the left-hand side of the Clock and Reset Tree window, Leda displays each flip-flop and latch controlled by the associated clock or reset, so that you can trace how changes in one portion of your design affect downstream sequential elements. You can double-click the (+) box icons to expand the display to different levels in the design hierarchy.

The Flipflops and Latches columns on the right-hand side of the window show the inferred hardware counts at each level of the design hierarchy.



Note

The Design Report tab on the Leda main window also shows totals for flip-flops and latches found in the design. In some cases, the number of latches reported in the Design Report is greater than the number shown in the Clock and Reset Tree windows. This is because some latches that are inferred in the design may not be clocked. The number of flip-flops reported in both places always matches.

When you click on objects in the schematic viewer, Leda highlights the corresponding element in the design hierarchy on the left-hand side of the window. For example, when you click on a flip-flop in the schematic viewer, Leda highlights that same flip-flop in the hierarchy view, so that you can identify and trace the clock that is driving it.

With these tabs active in the Path Viewer window, the elements in the Clock Tree and Reset Tree hierarchical display have these additional color-coded meanings:

- Green for clocks and resets
- Yellow for inferred hardware such as latches and flip-flops

Saving Error Reports

To save an Error Report in HTML format for later analysis, choose **Report > Save as HTML** from the main window. This brings up the “Save HTML report file as” window. Specify the name you want for the HTML file in the text field immediately below the displayed path (`leda.html` is the default).

Leda puts your HTML report in the current working directory by default. You can use the text field to navigate to a different location for your output file if you want. Then click on the OK button. This causes Leda to generate your Error Report and open the browser on the output file. You can link directly from the HTML Error Report to the HTML-based help for each rule that was violated to learn more about each issue.

In addition to information from the Error Report, Leda saves in HTML format the information about your user environment and configuration settings that you see in the Info Report tab. Leda uses a file named `file_info.html` to save this additional information.



Note

Leda also generates a directory named `file_name-html` in the current working directory. This directory contains other files that Leda uses to sort your results.

Post-processing Batch Mode Log Files

You can also use the Checker to view results stored in log files. You may have generated these log files using the Leda in batch mode (with the `-full_log` option) or using the Checker GUI. This mode of operation requires no project information. To open log files and have the results displayed in the Error Viewer, choose **Report > Open** from the Checker's main window.

The Checker GUI generates one log file in a directory named *project_name*-logs, even if one or more units or modules contain no errors. The batch-mode Checker generates a single log file named `leda.log` by default. You can specify a different name for this log file by using the `-l` switch when you run the Checker in batch mode. To use a single log file in append mode, use the `-lappend` switch instead.

When you generate the log file through the Leda GUI or using the batch mode with the `-full_log` switch, each message has the following syntax:

```
Nb : HDL_source_code
      ^^^^^^
HDL_File : Nb : Ruleset_Indicator >[ Severity_Level ] Rule_Label :
<Message>
#UNIT: HDL_LANG HDL_Library_Name> Unit_Name Full_HDL_File>
#RULE: Policy_Name Ruleset_Name Nb_Rule [ MASTER_Nb ]
#HTM1: [ HTML_Link_to_rationale ]
#HTM2: [ HTML_Link_to_user_documentation ]
#TRAK: HDL_File : nb [{, HDL_File : nb }]
```

A blank line separates messages. The last line gives a trace of the error message for chip-level rules. This information is used by the GUI to track the error message through the design hierarchy.



Attention

Error messages that do not have this format cannot be analyzed by the Error Viewer.

Table 14 summarizes the effects of using different Checker command-line switches on the content of the error messages printed on your screen and in the `leda.log` file when using the Checker in command-line mode.

The numbers correspond to the lines on a message. For example, if none of the options are present, the first three lines are printed to the screen and in the log files. Note that, to use the Error Report Viewer, you must first use the `-full_log` switch when you run the Checker in command-line mode.

Table 14: Command-Line Checker Error Report Options

Switches/Options	Default?	STDOUT	leda.log File
Null	Yes	First three lines	First three lines
<code>-old_format</code>	No	No effect	No effect
<code>-full_log</code>	No	No effect	All lines
<code>-nocode</code>	No	No effect	First two lines not present

Generating Leda Summary Information (Info Report)

In command-line mode, using the `-full_log` switch or the `-l logfile` option causes Leda to save the following kinds of summary information in a `leda.inf` file:

- Command-line options and switches you used when you invoked the Checker
- Information about your user environment in effect when you ran the Checker
- Configuration settings used by the Checker for that run
- Policy versions used and full paths to their locations

When you use the Checker from the GUI, Leda automatically saves this same information by default (see [“Checking Your Environment” on page 171](#)). In both cases, this summary information appears in the Info Report tab next to the GUI Error Viewer, when you later review your results.

If you run Leda in batch mode and want this summary information to appear at the beginning of the log file (`leda.log`), add the `-summary` switch to your batch invocation. This way, the log file you generate in batch mode has the same format that you get when you run a check in GUI mode.



Note

If you open two or more log files in the Error Report Viewer at the same time, Leda does not display the Info Report. This is to avoid merging information from two different Checker runs that may have used different environments or configurations.

Updating Projects

To update an existing project, follow these steps. For details on how to specify options, libraries, and source files, see [“Creating Projects to Check HDL Code” on page 91](#).

1. Choose **Project > Edit** from the main window. This brings up the Project Update Wizard (see [Figure 44](#)).

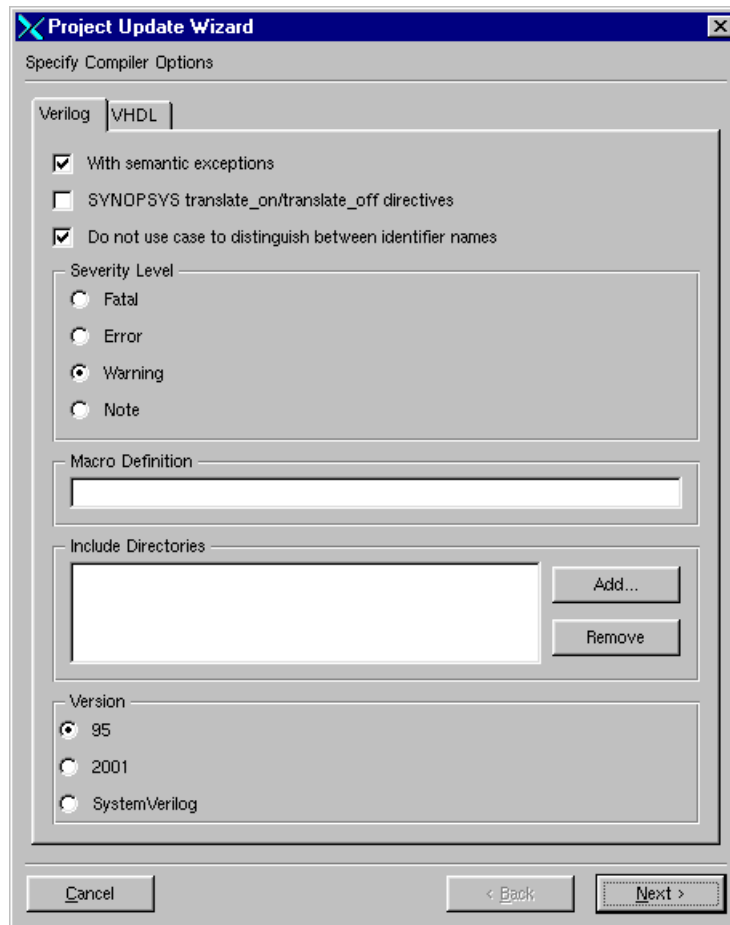


Figure 44: Project Update Wizard

The Project Update Wizard steps you through different windows to update your project:

1. Specify Compiler Options
2. Specify Libraries
3. Specify Source Files
4. Confirm and Create

These windows work the same way for updates as they do for creating new projects. You can step through each window to review and change your Leda projects as needed. Click on the Cancel button at any time if you are satisfied with your current project settings. Or click the Finish button from the last window in the Wizard to rebuild your projects with your updated settings in effect.

**Note**

For more information on setting libraries and resources, see [“Managing VHDL Libraries and Files”](#) on page 313.

5

Using the SDC Checker

Introduction

Integrated circuit designers use Synopsys Design Constraint (SDC) files to specify timing and area constraints for implementation and verification tools such as Design Compiler and PrimeTime. SDC uses a Tcl-based format. All commands in an SDC file must conform to Tcl syntax rules.

Leda Quality Checks

The Leda Constraint Checker verifies SDC quality by checking the following:

- Correctness of a SDC command
- Consistency between SDC commands
- Consistency between SDC and design
- Missing constraints

Leda Constraint Checker consists of ~150 pre-packaged SDC checks to verify quality of SDC file for RTL, pre-layout and post-layout stages of the design flow. Leda Constraint Checker has a programmable interface for you to write custom rules in tcl or C language. For more information, see the "[Leda Prebuilt Configurations](#)" on page 321

Top-versus-Block SDC Checks

Leda Constraint Checker verifies consistency between block level and top level design constraints files. Leda Constraint Checker verifies this consistency not by a command level comparison, but rather by verifying whether, effective constraints at block level are

contained in effective constraints at the top design level. Current pre-packaged checks to verify consistency between block & top level constraints covers the following SDC areas:

- Clocks
- Input & output delays
- Max/Min delays, false & multi-cycle paths

For more information, see the [“SDC-top-versus-block Prebuilt Configuration” on page 396](#)

SDC Equivalency Checks

Leda Constraint Checker verifies equivalency between two SDC files of a design. Leda Constraint Checker verifies the equivalency not by a command level comparison, but by verifying whether effective constraints of both SDC files are same or not. Current pre-packaged checks to verify the equivalency of two SDC files covers the following SDC areas:

- Clocks
- Input & output delays
- Max/Min delays, false & multi-cycle paths

For more information, see the [“SDC-equivalency Prebuilt Configuration” on page 397](#)

This chapter explains how to use Leda to identify problems with SDC files, in the following sections:

- [“Simplified Usage Model for SDC Checker” on page 135](#)
- [“Supported SDC File Tcl Commands” on page 137](#)
- [“Leda SDC Checker Tcl Commands” on page 140](#)
- [“Using a Tcl File For SDC Checks” on page 141](#)
- [“Defining Parameters for SDC Rules” on page 142](#)

Simplified Usage Model for SDC Checker

The Leda SDC Checker runs in Tcl Shell or batch mode. Basically, you read in and elaborate your design, read an SDC file, run the check, and report your results. Leda comes with a set of prepackaged rules for SDC checks (see the [Leda Constraints Rules Guide](#) for a detailed list).

You select which SDC rules you want to check using a configuration file that you point to using the `check -config` command. You can check as many or few rules as you want in one run using `rule_select` commands in your configuration file (see “[rule_select](#)” on page 239).

This usage model will suffice if you just want to check your SDC file for internal correctness and consistency with your design. Follow these steps:

1. Because all of the prepackaged SDC checker rules are organized in one policy, you can use a configuration file like this to select all of the rules in the SDC policy, as shown in the following configuration file example:

```
rule_deselect -all
rule_select -p CONSTRAINTS
```

2. Invoke Leda in Tcl Shell mode or batch mode. You cannot use GUI mode to run SDC checks:

```
% leda +tcl_shell
```

Note: You can also invoke the SDC checker in batch mode, using the following syntax:

```
% leda files -top top_name -constraint_file SDC_file_name -sdc
```

3. In Tcl shell mode, read in the HDL source files that you want to check:

```
leda> read_verilog path_to_file.v
-or-
leda> read_vhdl path_to_file.vhd
```

For more information, see “[read_verilog](#)” on page 296 or “[read_vhdl](#)” on page 299.

4. Elaborate the design by specifying the top-level module or entity:

```
leda> elaborate -top my_top_module -nohierdump
```

When you elaborate the design, this also clears the internal SDC database.

Note that the `-nohierdump` switch speeds up the tool. You only need to generate hierarchy if you want to later use the hierarchy browser in the GUI after your Checker run.

5. The SDC Checker reads environment variables only through the `read_constraints` command. You can set environment variables in a top-most Tcl file that sources the constraint files and pass this top-most Tcl file to the `read_constraints` command. For example, in the top-most Tcl file, you can set the `SYNTH` environment variable as follows:

```
set SYNTH [getenv SYNTH]
```

6. Read in SDC files in top-down order (that is, read files containing declarations before reading files that use those declarations). Read in an SDC file using the `read_constraints` command:

```
leda> read_constraints path_to_SDC_file
```

7. Read in the constraint file of a block using the `read_constraints` command:

```
leda> read_constraints -block instance_name path_to_block_SDC_file
```

8. Run the Checker using the `check` command with the `-sdc` switch and pointing to the configuration file using the `-config` option:

```
leda> check -sdc -config path_to_config_file
```

9. Report your results:

```
leda> report
```

Leda prints the results of your SDC Checker run on `STDOUT` and saves the information in a `leda.log` file. To see the results displayed in the GUI Error Viewer, choose **Report > Open** from the Checker's main window after running a check and load the `leda.log` file.

10. Exit the tool:

```
leda> quit
```



Note

You can also write your own custom SDC checker rules in Tcl or C/C++ using the supplied Constraint Query Language (CQL) APIs.

Supported SDC File Tcl Commands

Leda accepts SDC files you use with Design Compiler and PrimeTime, but parses and ignores application-specific SDC commands not needed for its checks. Leda supports the SDC commands listed [Table 15](#) and [Table 16](#).

Table 15: Supported SDC Design Constraint Commands

Information Types	Commands
Operating condition	set_operating_conditions
Wire load models	set_wire_load_min_block_size set_wire_load_mode set_wire_load_model set_wire_load_selection_group
System interface	set_drive set_driving_cell set_fanout_load set_input_transition set_load set_port_fanout_number
Design rule constraints	set_max_capacitance set_max_fanout set_max_transition set_min_capacitance set_min_fanout
Timing constraints	create_clock create_generated_clock set_gating_clock_check set_clock_latency set_clock_transition set_clock_uncertainty set_data_check set_disable_timing set_input_delay set_max_time_borrow set_output_delay set_propagated_clock set_resistance

Table 15: Supported SDC Design Constraint Commands (Continued)

Information Types	Commands
Timing exceptions	set_false_path set_max_delay set_min_delay set_multicycle_path
Area constraints	set_max_area
Power constraints	set_max_dynamic_power set_max_leakage_power
Porosity constraints	set_min_porosity
Logic assignments	set_case_analysis set_logic_dc set_logic_one set_logic_zero

Specifying Design Objects

Most of the constraint commands require a design object as a command argument. SDC supports both implicit and explicit object specification.

To avoid ambiguity, explicitly specify the object type using a nested object access command. For example, if you have a cell in the current instance named U1, the implicit specification is U1 and the explicit specification is [get_cells U1].

Table 16: Supported SDC Design Object Commands

Design Object	Access Command	Description
design	current_design	A container for cells. A block.
clock	get_clocks all_clocks	A clock in a design All clocks in a design
port	get_ports all_inputs all_outputs	An entry point to or exit point from a design All entry points to a design All exit points from a design
cell	get_cells	An instance of a design or library cell
pin	get_pins	An instance of a design port or library cell pin
net	get_nets	A connection between cell pins and design ports
library	get_libs	A container of library cells
lib_cell	get_lib_cells	A primitive logic element
lib_pin	get_lib_pins	An entry point to or exit point from a library cell

All SDC object names are case-insensitive to Leda. For example, these two commands are equivalent:

```
create_clock -name CK -period 1 -waveform { 0 5 } {ck1 ck2}
create_clock -name ck -period 1 -waveform { 0 5 } {ck1 ck2}
```

Handling Errors in SDC Files

When you read an SDC file into Leda using the `read_constraints` command, the SDC file parser issues error messages in the following cases:

- Syntax errors (for example, when a required option is missing)
- If an option requires only one signal but the corresponding command returns no signal or more than one signal. For example, the `-source` option below requires one signal, but the function `get_pins` returns two signals:

```
create_generated_clock -name CLK_BY_4 -source [get_pins {CLK}] \
    -divide_by 2 [get_pins {CLK_BY_4_reg/Q}]
```

Leda does not perform any semantic checks on SDC files.

Leda SDC Checker Tcl Commands

Leda supports a few special built-in Tcl commands for use with the SDC checker:

- Use the `read_constraints` command to read in an SDC file that you want to check. When you issue the `read_constraints` command, Leda reads the specified SDC file and stores the results in an internal database; it does not source the data like PrimeTime's `sdc_read` command. Read in SDC files in top-down order (that is, read files containing declarations before reading files that use those declarations).
- Use the `sdc_apply` command to apply the values for `set_case_analysis` commands to Leda's internal database. In order for this command to take effect you must next use the `propagate` command.
- Use the `propagate` command to propagate constants for signals defined with `set_case_analysis` commands in your SDC file.

In addition to these SDC-specific commands, you can use other general-purpose Leda Tcl commands in SDC checker Tcl scripts that you write for Leda (see [“Using a Tcl File For SDC Checks”](#) on page 141).



Note

For complete reference information on the built-in Tcl commands in Leda that you can use to configure rules, manage projects, and run checks, see [“Using Leda Tcl Shell Mode”](#) on page 187.

Using a Tcl File For SDC Checks

For more complicated SDC checks, you can read a Tcl file directly into Leda that contains SDC Checker commands. This is useful for stored procedures such as regression tests that need to be run over and over again. [Figure 45](#) shows an example Tcl file that you can use to run SDC checks in Leda.

```
set sdv_version 1.5
read_verilog <files>
elaborate -top TOP # cleans the SDC database

read_constraints file.sdc
sdc_apply -case_analysis
propagate -case_analysis
check -sdc -top TOP -config Blk_Test_RTL_Synth.tcl
report

read_constraints file2.sdc
sdc_apply -case_analysis
propagate -case_analysis
check -top B1 -sdc -config Blk_Func_RTL_Synth.tcl
report
check -top B1 -sdc -config Blk_Cross_Modes_RTL_Synth.tcl
report
```

Figure 45: Tcl File with SDC Checker Commands

Leda assumes that the SDC file version for your constraint files is 1.4. If your SDC files conform to a different version, use the `set sdv_version` command to specify the version in the first line of your Tcl script, as shown in this example.

Note the order dependency in this Tcl file. You must read the HDL source files and elaborate the design before you read an SDC file. This example uses the `read_verilog` command to read Verilog design files, but you can also use the `read_vhdl` command to read VHDL design files. For more information, see [“read_verilog” on page 296](#) or [“read_vhdl” on page 299](#).

The `sdc_apply` commands apply constants specified with `set_case_analysis` commands in the SDC file to the elaborated database.

The `propagate` commands propagate constant values specified in your SDC file using `set_case_analysis` commands.

In [Figure 45](#), we run three separate checks, using the `check` command with the `-sdc` switch. The first check:

```
check -sdc -top TOP -config Blk_Test_RTL_Synth.tcl
```

runs on the first SDC file read in (`file.sdc`). The `-config` option points to a configuration file (`Blk_Test_RTL_Synth.tcl`) that specifies the SDC rules that we want to check.

The second check:

```
check -top B1 -sdc -config Blk_Func_RTL_Synth.tcl
```

runs on the second SDC file read in (`file2.sdc`) and points to a different configuration file that specifies the set of rules we want to check.

The third check:

```
check -top B1 -sdc -config Blk_Cross_Modes_RTL_Synth.tcl
```

also runs on the second SDC file read in (`file2.sdc`) and points to a different configuration file that specifies the set of rules we want to check.

The `report` commands generate the check results on `STDOUT`.

To run this SDC checker script in Leda, use the following command:

```
% leda +tcl_file my_tcl_file
```

Defining Parameters for SDC Rules

The rules that you use to check SDC files for different modes and contexts must be parameterizable, so that you can define, for example, the names of the modes you want to compare and the names of the signals that you want to check in test and functional modes (for example). The [Leda Constraints Rules Guide](#) lists the current set of prepackaged SDC rules that you can use.

To see the parameters that are supported by any rule, use the `rule_get_parameter` command and specify the `rule_label`. For example:

```
leda> rule_get_parameter SDC_123
```

For more information about this command, see [“rule_get_parameter” on page 217](#).

Then define the value for a rule parameter using the `rule_set_parameter` command. For example:

```
leda> rule_set_parameter -rule SDC_123 -parameter MODE \
-value Test
```

You can set rule parameters interactively in the Tcl shell (as shown above), in the configuration file that you also use to specify which rules you want to check in a particular SDC Checker run, or in the Tcl script that you use to run SDC checks (see [“Using a Tcl File For SDC Checks” on page 141](#)).

6

Using Leda Batch Mode

Introduction

You can run the Leda Checker in batch mode by specifying switches and options on the command line when you invoke the tool. This way, the tool runs to completion unattended, which can be handy for script-driven test environments. This chapter explains how to use the Checker in batch mode, in the following major sections:

- [“Basic Usage Models and Rule Types” on page 145](#)
- [“Configuring the Checker” on page 146](#)
- [“Running Leda in Batch Mode” on page 148](#)
- [“Leda Batch Example Invocations” on page 162](#)
- [“Generating Projects in Batch Mode” on page 163](#)
- [“Checker Batch Mode Results” on page 166](#)

Basic Usage Models and Rule Types

There are two basic ways to run the Checker in batch mode:

- **Method 1**—Pass a list of HDL files directly on the command line, with no switches. This causes the Checker to first analyze or compile the code before checking all units or modules contained in the specified HDL source files.
- **Method 2**—Use the `-o` option with the name of an HDL library or the name of an HDL library and a unit already compiled into that library. With this approach, the Checker skips the compilation phase. If you do not specify any units, the Checker works on all units in the library by default.

Rules can be classified into four major families:

- **Block-level Rules**—unit-wide (for example, detect multiple clocks in an architecture).
- **Chip-level Rules**—design-wide (for example, all flip-flops in the design must be active on the rising edge).
- **Netlist Rules**—rules that run on the gate-level netlist for the design (for example, control signal crossing clock domain without data transfer).
- **SDC Rules**—rules that check Synopsys Design Constraint (SDC) files for internal consistency and consistency with the design that they constraint.

All checks are on by default. You don't need to use the `-block`, `-chip`, `-netlist`, or `-sdc` switches to enable these checks unless you want to run just one type of check. For example, if you want to run chip-level checks only, use the `-chip` switch on the command line. This enables chip-level checks and disables all other types of checks. Or, if you want to run just block- and chip-level checks, use the `-block` and `-chip` switches on the command line. This enables block- and chip-level checks and disables all other types of checks.

Configuring the Checker

First, set up your environment for running the Checker (see [“Leda Environment Variables” on page 317](#)). Next, define any macros needed for expanding rules prior to checking, as explained in [“Defining Macro Values for Rules” on page 82](#).

If you want to propagate constants in the Checker, enter `set_case_analysis` commands in a constraint file, and point Leda to the file using the `-case_analysis_file` option (for more information, see [“Propagating Constants” on page 96](#)).

Using plibs to Set Library Logical/Physical Mapping

When executing the Checker, you can specify the name of the library where you want the results of the check to be stored using the `-work` switch. You must map this library's logical name to a physical location (file or directory). You do this mapping using the `plibs` file. If no mapping exists, Leda uses the current working directory as the physical location for storing results of the check.

In addition, working libraries must be able to find the physical locations of any resource libraries used (for example, GTECH, VITAL). You can specify the physical location of these resource libraries in a plibs file. The syntax of the plibs file is simple:

Library_Logical_Name */library/physical/location*

For example:

IEEE **\$LEDA_PATH/resources/resource_93/IEEE**

In this example, Leda evaluates the \$LEDA_PATH variable at compile time, so you can move your resource libraries anywhere you want on your network. The Checker looks for two plibs files: a global plibs file and a local plibs file. The global plibs file is located in the \$LEDA_PATH/resources/resource_93 (or resource_87) directory or, if you have \$LEDA_RESOURCES defined, in \$LEDA_RESOURCES/resource_93 (or resource_87). You can use this mechanism to store the mappings of VHDL resource libraries that you always require (for example, STD, IEEE, SYNOPSIS...).

Put the local plibs file in either the current working directory or in \$HOME. If Leda cannot find a local plibs file, it creates the physical directory corresponding to the library specified with the `-work` switch in the current working directory. If you don't specify the `-work` switch on the command line, Leda creates a library named `.leda_work` in the current working directory by default, unless there is an explicit mapping for the logical name `WORK` in your local plibs file. You can overwrite the mappings specified in the global plibs file by specifying different mappings in the local plibs file.

Running Leda in Batch Mode

Invoke Leda in batch mode as shown in the following example:

```
% $LEDA_PATH/bin/leda [list-of-options]
```

If you do not specify any options or you only specify the `-project` option, the Checker GUI comes up. Depending on the command-line options and the rules to be checked, the tool may elaborate your HDL code. This happens only if you have chip-level rules selected and you activate chip-level checking. For VHDL files, make sure you specify them in the correct compilation order.

The options that you can specify on the command line are sorted into three different categories: VHDL-only, Verilog-only, and common. The common options can be used on both VHDL and Verilog files. Following are detailed descriptions of the command-line options:

- [“Common Command-Line Options and Switches” on page 148](#)
- [“VHDL Command-Line Options” on page 157](#)
- [“Verilog Command-Line Options” on page 159](#)

Common Command-Line Options and Switches

The options that you can use on both Verilog or VHDL files are described in [Table 17](#).

Table 17: Common Command-Line Options and Switches

Option/Switch	Description
-b	<p>Use the <code>-b</code> switch to make the Checker create one log file per unit checked. The name of each file is <code>leda_unit_name.log</code>. Log files are opened in write mode, thus overwriting the previous contents.</p> <p>This switch also causes Leda to save information about your user environment and Leda configuration in a file named <code>leda.inf</code> in the same directory as the log file. If you want to specify a different destination directory for the <code>leda.inf</code> file, use the <code>-log_dir</code> option instead.</p> <p>If you use both the <code>-b</code> and <code>-summary</code> options in the same batch command, Leda prints the summary information in a separate <code>leda.log</code> file.</p> <p>For information on <code>leda.inf</code> files and how to use them, see “Post-processing Batch Mode Log Files” on page 128.</p>
-blast	<p>Bit blasting of vectors or buses is on by default starting with version 4.1.</p>

Table 17: Common Command-Line Options and Switches (Continued)

Option/Switch	Description
-block	The Checker executes block-level checks by default as long as you have some block-level rules selected. You can also use this switch to run block-level checks only and disable all other types of checks, regardless of the types of rules you have selected in your configuration file.
-c	Use the -c switch if you just want to compile your HDL source files and not check them. When you use this switch, the Checker only analyzes the code for compatibility with VHDL or Verilog syntax and semantics.
-case_analysis_file	Use the -case_analysis_file option to point Leda to an ASCII text file containing set_case_analysis commands that specify constant values for primary inputs or internal signals. For more information, see “Propagating Constants” on page 96 .
-chip	The Checker executes chip-level checks by default as long as you have some chip-level rules selected and use the -top option to specify the top-level unit in your design. You can also use this switch to run chip-level checks only and disable all other types of checks, regardless of the types of rules you have selected in your configuration file.
-clock_file	Use the -clock_file option to specify the synchronous clocks in the design with set_clock_groups command. The checker uses this information for doinf chip-level and netlist checks. For more information, see “Clock Grouping Feature” on page 66 .
-clockdump	In -full_log mode, use the -clockdump switch to enable use of the Clock and Reset Tree browsers when you load a log file into the Error Viewer after checking your design. Note that this switch can slow Leda’s performance when checking large netlists. See “Using the Clock and Reset Tree Browsers” on page 126 .
-config <i>full_path_to_file</i>	Use the -config option to point Leda to a configuration file containing a rule configuration that you want to use. The rule configuration you specify with this option takes priority over the one specified with the LEDA_CONFIG environment variable. You can use this option to specify one of the prebuilt configurations (RTL, Gate-level, Leda-classic, Leda-optimized). In this case, you don’t need to specify the full path, but the configuration names are case-sensitive, and need to be typed exactly as shown. For more information on specifying rule configurations for the Checker, see “Using Prebuilt Configurations” on page 99 .

Table 17: Common Command-Line Options and Switches (Continued)

Option/Switch	Description
-config_summary	Use the -config_summary option to print the configuration summary on the console. The summary is displayed after the rules checking is done and is saved to \$PWD/leda_config.log.
-constraint_file <i>file</i>	For SDC checks, use the -constraint_file option to point to the name of the SDC file to read in to the Checker (see “Using the SDC Checker” on page 133). This option activates all the checkers as well as the SDC checker.
+exec+ <i>rule_file.ext</i> + <i>function</i>	<p>Use the +exec+ option to test a netlist checker rule developed in C in the form of an object file or shared library, where <i>rule_file.ext</i> is the object or shared library file that implements the rule and <i>function</i> is the C function name for a rule defined in the <i>rule_file.ext</i>. Do not include the “rule”_ prefix for your C function when you specify the function name on the command line. For example, if your C function is named rule_B6000, specify the <i>function</i> as B6000.</p> <p>The <i>rule_file</i> name extension (<i>ext</i>) is platform-dependent:</p> <ul style="list-style-type: none"> Solaris—<i>rule_file.o</i> Linux—<i>rule_file.so</i> HP-UX—<i>rule_file.sl</i> <p>For more information on Leda’s C interface for writing custom netlist checking rules, see the Leda C Interface Guide.</p>
-forecehierdump	Use the -forcehierdump switch to force creation of the full hierarchy browser database. By default, Leda creates hierarchy browser data only for the first 1,000 instantiations in a module. You can set a different number for the maximum number of instantiations using the -maxhierdump <i>N</i> option. To disable creation of hierarchy browser data, use the -nohierdump switch.
-full_inf	Use the -full_inf switch to enable the display of deactivated rules and the violation summary in the .inf file

Table 17: Common Command-Line Options and Switches (Continued)

Option/Switch	Description
-full_log	<p>Use the -full_log switch if you want error messages printed to the log files in full format, which looks like this:</p> <pre> 1 nb : HDL_source 2 ^^^^ 3 HDL_file : nb : ruleset [severity_level] label: message 4 #UNIT: HDL_library HDL_unit FULL_HDL_file 5 #RULE: policy ruleset rule_file nb_rule [MASTER_Nb] 6 #HTM1: [address] 7 #HTM2: [address] 8 [#TRAK: HDL_file : nb [{,HDL_file : nb}]] </pre> <p>Note: You must use the -full_log switch if you want to later use the Error Report Viewer in the Checker GUI to review the error messages. For more information, see “Post-processing Batch Mode Log Files” on page 128.</p>
+gui	Use the +gui option to open the GUI.
-h	Prints the tool’s help information.
-html <i>html_report_name</i>	Use the -html option to make the Checker generate an error report in HTML format. Use the <i>html_report_name</i> argument to specify the name of the report file. Note that when you use this option, you get the -full_log format by default.
-ignore_rule_pragmas	Use the -ignore_rule_pragmas option to disable the leda off/on pragmas while running the checker.
-l <i>logfile</i>	Use the -l option to specify the name of the log file where you want messages to be stored. If you don’t use this option or the -b or -nolog switches, Leda uses the default log file name of leda.log.
-lappend <i>logfile</i>	Does the same thing as the -l option, but opens the log file in append mode. When you use this option, Leda does not create a leda.inf file containing information about your user environment and Leda configuration. In addition, if there is a pre-existing leda.inf file created during a previous run of the Checker, Leda removes it, because it is no longer valid.
-log_dir <i>directory</i>	Use this option, in combination with the -full_log switch, to specify a destination directory other than the default of the current working directory for the <i>logname.inf</i> file, where Leda saves information about your user environment and Leda configuration when you run the Checker.

Table 17: Common Command-Line Options and Switches (Continued)

Option/Switch	Description
-maxhierdump <i>N</i>	Enables creation of the hierarchy browser database if there are less than the specified number (<i>N</i>) of instantiations in a module. The default is 1,000.
-maxmessages <i>N</i>	Use the -maxmessages option to set the maximum number (<i>N</i>) of messages per selected rule (per language) that Leda flags. The default is 100. If you want Leda to report all violations regardless of the number found, use the -nomaxmessages N switch.
-maxviolations <i>N</i>	Use the -maxviolations option to set the maximum number (<i>N</i>) of violations per selected rule that Leda flags. The default is 100. If you want Leda to report all violations regardless of the number found, use the -nomaxviolations switch.
-netlist	The Checker executes netlist checks by default as long as you have some netlist rules from the Design policy or custom netlist rules that you wrote selected and use the -top switch to specify the top-level unit in your design. You can also use this switch to run netlist checks only and disable all other types of checks, regardless of the types of rules you have selected in your configuration file.
-nobanner	Use this switch to stop the Checker from printing a banner.
-noclockdump	Disables generation of data for the GUI Clock and Reset Tree browsers. The Clock and Reset Tree browsers are enabled by default.
-nocode	Use the -nocode switch to reformat log file error messages. This masks the HDL line in error messages printed to the screen and to the log files. The messages have the following format: <pre>test.vhd:19: [ERROR] R_552_1: Signal is not assigned on all branches and may infer a latch</pre> If you combine the -nocode switch with the -full_log switch, Leda ignores the -nocode switch.
-nocompilemessage	Use the -nocompilemessage switch if you don't want Leda to print module information during compile phase.
-noecho	Use the -noecho switch to suppress warning messages printed when: <ul style="list-style-type: none"> • You check chip-level rules, but you do not specify any top unit. • You activate chip-level checks, but no chip-level rules exist. • Chip-level rules exist within the policy to be checked, but you do not activate chip-level checks.

Table 17: Common Command-Line Options and Switches (Continued)

Option/Switch	Description
-nohierdump	Use the -nohierdump switch to turn off generation of the hierarchy browsing database. This speeds up tool performance, but disables the hierarchy browser in the GUI after a Checker run.
-nolog	Use the -nolog switch to stop the Checker from generating log files.
-nomaxmessages <i>N</i>	Use the -nomaxmessages switch if you don't want Leda to limit the number of messages per rule (per language). If you want Leda to report all violations regardless of the number found, use the -maxmessages N switch.
-nomaxviolations	Use the -nomaxviolations switch if you don't want Leda to limit the number of violations per rule. The default is 100 violations per rule. You can also set this to a different number using the -maxviolations N option.
-nowarning	Use the -nowarning switch to suppresses warning messages generated during compilation and elaboration.
-o <i>LIB</i> [<i>UNIT</i>]	Use the -o option to specify the library or unit to be checked. For example, the command: <pre>% leda -o LIB UNIT -p rmm_rtl_coding_guidelines</pre> checks unit <i>UNIT</i> from library <i>LIB</i> . You must have already compiled this unit. If no units are specified on the command line, all units in the library are checked.
-p <i>policy</i>	Use the -p option to specify the names of policies to check. If followed by one or more -r options, Leda checks only the rulesets specified by the -r options. If there are no -r options, Leda checks all rulesets. Here are some examples: <pre>% leda -p rmm_rtl_coding_guidelines % leda -p rmm_rtl_coding_guidelines -r coding_for_synthesis % leda -p rmm_rtl_coding_guidelines -p ieee_rtl_synth_subset % leda -p rmm_rtl_coding_guidelines -r coding_for_synthesis % leda -p ieee_rtl_synth_subset</pre>
-project <i>project</i>	Use the -project option to specify the name of a project containing your HDL source files that can be opened in the GUI. Note that the project must be a simple name, not the full path. When you use this option, Leda stores information about your user environment and Leda configuration in a file named <i>project.inf</i> , instead of <i>leda.inf</i> . For information on .inf files and how to use them, see “Post-processing Batch Mode Log Files” on page 128.

Table 17: Common Command-Line Options and Switches (Continued)

Option/Switch	Description
-quiet	Use the -quiet switch to turn off printing of error messages and log file entries to STDOUT.
-r <i>ruleset</i>	Use the -r option to specify the rulesets to be checked from a given policy. To find the ruleset names in a given policy, see the corresponding policy documentation in the <i>Leda Prepackaged Rules Guides</i> located in the \$LEDA_PATH/doc directory. Note that you must precede every ruleset with a separate -r option.
-sdc	The Checker executes SDC checks by default as long as you have some SDC rules selected in your configuration. You can also use this switch to run SDC checks only and disable all other types of checks, regardless of the types of rules you have selected in your configuration file. See “Using the SDC Checker” on page 133 .
-severity <i>severity</i>	<p>Use the -severity option to specify the lowest severity for which you want the Checker to print messages. Messages with a severity below the specified value are not printed. The allowed values, in order of importance, are:</p> <ul style="list-style-type: none"> • NOTE (default) • WARNING • ERROR • FATAL <p>For example, if you specify:</p> <pre style="text-align: center;">-severity ERROR</pre> <p>the Checker only prints messages with a severity of ERROR or FATAL.</p>
-sort <i>sort_by</i>	<p>Use the -sort option to specify how you want rule violations sorted in the leda.log file. The default sort is by rule. Use <i>sort_by</i> to specify a different sorting option. The legal values are:</p> <ul style="list-style-type: none"> • label • policy • severity • file • language • module • master_id <p>Note: If you use -sort, you must also specify full_log.</p>

Table 17: Common Command-Line Options and Switches (Continued)

Option/Switch	Description
-summary	Use the -summary option if you want summary information to appear at the beginning of the leda.log file. Summary information is already available in the GUI Error Viewer (Info Tab) if you specify -full_log as part of your batch invocation. The -summary switch puts the same information at the beginning of your log file. This way, the log file format you get when running in batch mode matches the log file format you get in GUI mode.
-test_asynch <i>RST</i>	Use the -test_asynch option to specify the test reset signal <i>RST</i> and indicate that it is active on “1” and has a hold value of “0” during the scan shift phase. With RTLDRCC©, this corresponds to the following command: <pre>set_signal_type test_asynch RST</pre>
-test_asynch_inverted <i>RST</i>	Use the -test_asynch_inverted option to specify the test reset signal <i>RST</i> and specify that it is active on “0” and has a hold value of “1” during the scan shift phase. With RTLDRCC©, this corresponds to the following command: <pre>set_signal_type test_asynch_inverted RST</pre>
-test_clk_falling <i>CLK</i>	Use the -test_clk_falling option to specify test clock signal <i>CLK</i> and specify that the first edge in this clock’s cycle is the falling edge. With RTLDRCC©, this corresponds to the following command: <pre>create_test_clk CLK -w{N1 N1-N2}</pre> <p>In Leda’s DFT checks, no delays are taken into account. Leda always assumes that the test clock period is 100 ns and the strobe point occurs at 95 ns (default RTLDRCC© values). Leda also assumes that all test clock events occur before this strobe point.</p>
-test_clk_rising <i>CLK</i>	Use the -test_clk_rising option to specify test clock signal <i>CLK</i> and specify that the first edge in this clock’s cycle is the rising edge. With RTLDRCC©, this corresponds to the following command: <pre>create_test_clk CLK -w{N1 N1+N2}</pre> <p>In Leda’s DFT checks, no delays are taken into account. Leda always assumes that the test clock period is 100 ns and the strobe point occurs at 95 ns (default RTLDRCC© values). Leda also assumes that all test clock events occur before this strobe point.</p>
+tcl_file <i>script.tcl</i>	Use the +tcl_file option to execute a Tcl script that contains Leda Tcl commands.

Table 17: Common Command-Line Options and Switches (Continued)

Option/Switch	Description
+tcl_rule+ <i>file.tcl</i> + <i>procedure</i>	Use the +tcl_rule+ option to test a Tcl-based design netlist rule, where <i>file.tcl</i> contains a procedure named <i>rule_label</i> . Do not include the “rule”_ prefix for your Tcl procedure when you specify the procedure name on the command line. For example, if your Tcl procedure is named rule_B6000, specify the <i>procedure</i> as B6000.
+tcl_shell	Use the +tcl_shell switch to make Leda enter Tcl shell mode. To enable DQL design queries in the Tcl shell, you must elaborate the design in batch mode or use the elaborate command in the Tcl shell mode (see “ elaborate ” on page 283).
-top <i>UNIT</i>	Use the -top option to specify the top unit or module of your design hierarchy. This is required in order to check for chip-level rule violations.
-translate_directive	Use the -translate directive switch if you don’t want HDL source code that falls between the following Synopsys directives to be compiled: <ul style="list-style-type: none"> • synthesis_off and synthesis_on • translate_off and translate_on
-upgrade400	Use the -upgrade400 to translate rule deactivation commands in.leda_select files into equivalent Tcl commands. Don’t use any other command-line options when you run the translator. For more information, see “ Translating .leda_select Files ” on page 106.
-version	Prints the current version of the Checker.
-work <i>LIB</i>	Use the -work option to specify the name of the library into which all files will be analyzed. This option is ignored if you do not specify any files on the command line. You specify the physical location of the specified library in a file called plibs (see “ Using plibs to Set Library Logical/Physical Mapping ” on page 146). If you do not specify the -work option, Leda analyzes the plibs file to see if there is a physical library mapped to the logical name WORK. If not, the library .leda_work is used. If the plibs file contains no physical location for this library, the Checker creates it locally.

VHDL Command-Line Options

The options that you can use only on VHDL files are described in [Table 18](#).

Table 18: VHDL Command-Line Options and Switches

Option/Switch	Description
-files <i>filename</i>	<p>Use the -files option to specify that all VHDL files to be analyzed and checked are listed in the text file <i>filename</i>. If you use this option in conjunction with the -project option, a #Files or #Dirs clause in the file indicated by the <i>filename</i> argument must contain the library name.</p> <p>If you want a file or directory to be treated as a resource library, and therefore excluded from Leda block-level and chip-level checks, add the nochecklib argument as shown in the following examples:</p> <pre>#Files <libname> nochecklib #Dirs <libname> nochecklib</pre> <p>If you don't specify nochecklib, the default is checklib. See "Example for -files" at the end of this table.</p>
-lang <i>LANG</i>	<p>Use the -lang option to select the mode to use when analyzing code. This option can take one of the following values for <i>LANG</i>:</p> <ul style="list-style-type: none"> • 87—analyzed using VHDL 87 syntax and semantics. • 87e—analyzed using VHDL 87 syntax and semantics, with some semantic exceptions. • 93—analyzed using VHDL 93 syntax and semantics. • 93e—analyzed using VHDL 93 syntax and semantics, with some semantic exceptions. This is the default. <p>For information on semantic exceptions and how to control their use, see "Writing and Checking HDL Designs" on page 51.</p>
-mk	<p>Use the -mk switch to make the Checker automatically deduce the compilation order for your VHDL source files.</p>
-mkk	<p>This switch works alike -mk switch, but continues even if there is a syntax error.</p>
+nochecklib	<p>Use the +nochecklib option on the command line to specify VHDL resource libraries that you don't want Leda to check for errors. Leda's default behavior is to check all libraries passed in on the command line.</p> <p>Note that if you use +nochecklib, you must also use the -work option to specify the name of the library into which all files will be analyzed (see "-work LIB" on page 114).</p> <p>The -files and +nochecklib options are mutually exclusive. You cannot use both options on the same command line. If both options appear, Leda ignores the +nochecklib option and determines which libraries to check based on the contents of the file specified by the -file option.</p>

Example for -files

You can use more than one #Files or #Dirs clause in *filename* to specify more than one library, but you cannot specify any library more than once. For example, a *filename* with the following contents causes Leda to analyze the specified libraries correctly:

```
#Files lib1
source/lib1_cell11.vhdl
source/lib1_cell12.vhdl
source/lib1_cell13.vhdl
#Files lib2
source/lib2_cell11.vhdl
source/lib2_cell12.vhdl
#Files lib3
source/lib3_cell11.vhdl
```

But this next example causes Leda to analyze the files incorrectly:

```
#Files lib1
source/lib1_cell11.vhdl
#Files lib2
source/lib2_cell11.vhdl
#Files lib3
source/lib3_cell11.vhdl
#Files lib1
source/lib1_cell12.vhdl
#Files lib2
source/lib2_cell12.vhdl
#Files lib1
source/lib1_cell13.vhdl
```

The highlighted portions of this example are incorrect because the lib1 and lib2 libraries are declared more than once. If you list the same library more than once and use nochecklib in one of them, Leda uses the library explicitly declared as nochecklib. If you specify the same library more than once and use conflicting nochecklib and checklib arguments in them, Leda issues an error message.

Verilog Command-Line Options

The options that you can use only on Verilog files are described in [Table 19](#).

Table 19: Verilog Command-line Options and Switches

Option/Switch	Description
-a	This is a simulator accelerator switch. It is ignored by Leda.
+checklib+<libname>	<p>If <i>libname</i> refers to a directory specified with -y, Leda includes all modules found in that directory for both block-level and chip-level checks. For example:</p> <pre>% leda files.v -y /path/to/design/dir1 -y /path/to/design/dir2 +checklib+/path/to/design/dir2</pre> <p>For this command line, Leda checks all the modules in dir2, but not the modules in dir1 (see -y library_dir on page 161).</p> <p>If <i>libname</i> refers to a file specified with -v, Leda includes all modules found in that file for both block-level and chip-level checks. For example:</p> <pre>% leda files.v -v lib1.v -v lib2.v +checklib+lib2.v</pre> <p>For this command line, Leda checks all the modules in lib2.v, but not the modules in lib1.v (see -v library_file on page 161).</p> <p>Note: For chip-level rules, Leda does not flag errors contained completely in modules not be checked. But if any trace element of the error is in a module to be checked, Leda flags the error.</p>
-d	This is a simulator decompile switch. It is ignored by Leda.
+define+macro [=val]	Use the +define argument to define the <i>macro</i> macro and assign <i>val</i> to it.
-f <i>filename</i>	Use the -f option to specify a command file that can list Verilog files and any other options that you want to specify.
-i <i>filename</i>	Use the -i option to specify an interactive file for the simulator. It is ignored by Leda.
+incdir{+directory}	Use the +incdir argument to specify the directories to be searched for included files.
-k <i>filename</i>	Use the -k option to specify a key file for the simulator. It is ignored by Leda.

Table 19: Verilog Command-line Options and Switches (Continued)

Option/Switch	Description
-lang <i>LANG</i>	<p>Use the -lang option to select the mode to use when analyzing code. This option can take one of the following values for <i>LANG</i>:</p> <ul style="list-style-type: none"> • 95—analyzed using syntax and semantics specified in the Verilog LRM. • 95e—analyzed as Verilog 95, but with some commonly used semantic exceptions. Emulates analyzers that do not conform to the Verilog LRM. <p>For information on semantic exceptions and how to control their use, see “Writing and Checking HDL Designs” on page 51.</p>
+libext{+.string}	Use the +libext argument to specify file extensions for files in library directories (see option -y). You can only use one libext clause on the command line. The default file extensions for option -y are .v and .V.
+max_case+<val>	Use the +max_case option to specify the maximum width of a case expression in a case statement. The default value is 8.
+max_casexz+<val>	Use the +max_casexz option to specify the maximum width of a case expression in a casex/casexz statement. The default value is 8.
-q	Use the -q switch to specify a simulator quiet option. It is ignored by Leda.
-s	Use the -s switch to specify a simulator stop option. It is ignored by Leda.
+sv	Use the +sv switch to make Leda parse and check language compliance for SystemVerilog.
-sverilog	Use the -sverilog switch to make Leda parse and check language compliance for SystemVerilog. This works the same way as +sv, but is present for compatibility with the VCS command line.
-t	Use the -t switch to specify a simulator trace generation option. It is ignored by Leda.
-u	<p>Use the -u switch to make Verilog analysis case-insensitive. This changes all characters in identifiers to uppercase.</p> <p>Note: When you use the -u switch, the Verilog language itself becomes case-insensitive, but not the mechanism Leda uses to find library directories and files specified with the -y or -v options or used during elaboration.</p>
-uselrmsize	Forces width of integers to be evaluated to 32 bits as per LRM. Also added to check and run Tcl commands. This option will handle non-size constants (integer and 'b0) as 32 bits width.

Table 19: Verilog Command-line Options and Switches (Continued)

Option/Switch	Description
-use_netlist_reader	Use the -use_netlist_reader option to invoke netlist reader.
-usev2klrmsize	Use the -usev2klrmsize switch to strictly apply v2k LRM bit widths.
+v2k	Use the +v2k switch to make Leda parse and check language compliance for supported Verilog 2001 constructs. For information on current supported constructs, see “Verilog 2001 Support” on page 65 . Note: This is the same switch used with the Synopsys VCS simulator.
-v <i>library_file</i>	Use the -v option to specify a library file. The Checker scans each library file for module definitions and tries to resolve all unresolved module instances in the Verilog source files. Note: This option works just like the VCS -v option, except that Leda does not check modules coming from files specified after -v unless you also use the +checklib option (see +checklib+<libname> on page 159).
-w	Use the -w switch to suppress Checker messages with severities lower than ERROR.
-x	Use the -x switch to specify a simulator vector net expansion option. It is ignored by Leda.
-y <i>library_dir</i>	Use the -y option to specify a library directory that contains Verilog source files. The Checker scans the files in each library directory for module declarations and tries to resolve all unresolved module instances in the Verilog source files. This option work for files containing more than one module. Note: This option works just like the VCS -y option, except that Leda does not check modules coming from files specified after -y unless you also use the +checklib option (see +checklib+<libname> on page 159).

Leda Batch Example Invocations

To see how to use Leda in batch mode, you can use the HDL source files located in `$LEDA_PATH/test/mixed/work/src`. In this example, you can analyze these files and store the results in the same location. This is more important for VHDL files; the VHDL term “library” is used here to represent this location. The name of the library is `Leda_WORK`. You specify the physical location of `Leda_WORK` in a plibs file (see [“Using plibs to Set Library Logical/Physical Mapping” on page 146](#)). When compiling the source code, follow these guidelines:

- Compile Verilog code first and compile all Verilog code together. To do this, use standard Verilog batch options (such as `-f` and `+incdir+`). For details on Verilog batch options, see [“Verilog Command-Line Options” on page 159](#).
- Compile VHDL code in the correct compilation order. Otherwise, it will not compile successfully. To do this, use standard VHDL batch options (such as `-files` and `-mk`). For details on VHDL batch options, see [“VHDL Command-Line Options” on page 157](#).
- Make sure that all your working libraries are listed in the project’s plibs file.

You can build the mixed-language example project using the following commands:

```
% leda -c -work Leda_WORK $LEDA_PATH/test/mixed/src/*.v
% leda -c -work Leda_WORK $LEDA_PATH/test/mixed/src/misc_logic.vhd
% leda -c -work Leda_WORK $LEDA_PATH/test/mixed/src/stage1.vhd
```

Keep the following points in mind when you build a project:

- The `-c` switch tells the Checker to perform analysis only and not to check any rules.
- To run chip-level checks, you must indicate the top unit in the design so that the chip-level Checker can follow connectivity paths from this unit. In this example, the top unit is the Verilog module named “top.”
- To indicate what rules to check, you use the `-p` and `-r` switches. However, in this example, you are checking the prepackaged policies only and therefore do not need to use these switches.
- To execute the prepackaged policies with both block-level and chip-level rules on, type the following:

```
% leda -o Leda_WORK -top top -full_log
```

- The `-full_log` switch tells the Checker to generate a log file named `leda.log` that you can later review using the interactive Error Report Viewer in the Checker GUI.

- You can also execute block-level checks as the project is being built. To do this, remove the `-c` switches from the command lines in the previous examples, and add the `-full-log` switch if you want to later analyze the results in the Error Report Viewer.
- If you want to generate the log file in HTML format, use the `-html` switch.

Generating Log Files in Batch Mode

When you run a design without creating a project in the batch mode (see command below):

```
% leda -top <top_name> *.v -config config.tcl
```

then, files `leda.log` and `leda.inf` are created in the present working directory.

When you create a project in the batch mode by executing command

```
% leda -top <top_name> *.v -config config.tcl -project leda.pro
```

then, a directory `leda-logs` is created in the present working directory. Files `leda.log` and `leda.inf` are created in this directory.

Generating Projects in Batch Mode

You can generate a project in batch mode and read the log file later in the GUI. You can do this for Verilog-only, VHDL-only, and mixed-language projects. The advantage of this approach is that Leda can run unattended in batch mode at any time (even during off-peak hours). You can then use the GUI for viewing the results and making minor changes. For information on viewing log files in the Checker's Error Report Viewer, see [“Post-processing Batch Mode Log Files” on page 128](#).

Verilog-only Projects

Generating Verilog-only projects is straightforward. Just use the `-project` switch, as shown in the following example:

```
% leda -work Leda_WORK $LEDA_PATH/test/mixed/src/*.v -project ver
```

You can also use the `-f` switch to pass file information to Leda's project creation routine.

VHDL-only Projects

VHDL-only projects are slightly more complicated because you have to be careful about the compilation order for your source files. VHDL projects also frequently make use of more than one library. To make this work, you have three different options, as shown in the following examples:

- Use the `-project` switch to specify a project name
- Use the `-mk` switch to make Leda deduce the compilation order
- Use the `-files` option to pass file information to Leda's project creation routine

The data file you supply with the `-files` option allows you to easily specify large projects on the command line. It also allows you to specify into which library a given file is compiled, as shown in the following syntax descriptions:

```
file_contents ::= { [list_of_files] | [list_of_dirs] | [comments] }
list_of_files ::= #Files [<LIB>]
                 file_name | comment {file_name | comment}

file_name ::= <File Name>

list_of_dirs ::= #Dirs [<LIB>]
               dir_name | comment {dir_name | comment}

dir_name ::= <Directory Name>

comment ::= --<text>
```

Keep the following points in mind when building VHDL projects in batch mode:

- You can alternate the *list_of_files*, *list_of_dirs*, or *comment* rules throughout the file.
- The keywords Files and Dirs are not case-sensitive.
- If you do not specify any libraries on the command line or in the data file, `.leda_work` is assumed by default.
- If you do not specify any library for some or all sections of the data file, but you specify a library with the `-work` switch on the command line, this library is assumed to be the default library.
- If you do not specify any library on the command line, then all sections of the data file must have an explicit library.
- The library names given in the data file must be the logical names of the libraries. You can specify physical mappings in the `plibs` file.
- Separate file names by new lines.

- Comments begin with the characters “--” and continue to the end of the line. Comments can appear anywhere in the file, and stop at the end of the current line (as in VHDL).

For example, the following file contents indicate that all files go into a directory called `Leda_WORK`:

```
#Files Leda_WORK
$LEDA_PATH/test/mixed/src/misc_logic.vhd
$LEDA_PATH/test/mixed/src/stage1_vhd.vhd
```

Or, you could specify:

```
#Dirs Leda_WORK
$LEDA_PATH/test/mixed/src
```

The batch syntax is then:

```
% leda -files f1.dat -project vhd
```

Or, if you're not sure of the compilation order, you can specify:

```
% leda -files f1.dat -project vhd -mk
```

Mixed-Language Projects

You can create a mixed VHDL/Verilog project by combining the information in the previous sections. For example, the following command line creates a mixed VHDL/Verilog project:

```
% leda -work Leda_WORK $LEDA_PATH/test/mixed/src/*.v -files f1.dat \
  -mk -project mixed
```

Checker Batch Mode Results

After you run the Checker in batch mode, Leda returns a status, creates a log file of your results, and records the environment used for that run of the Checker, as explained in the following sections:

- [“Checker Return Status” on page 166](#)
- [“Viewing Checker Results” on page 167](#)
- [“Checking the Environment” on page 167](#)

Checker Return Status

When the Checker terminates, it returns a completion status in the `$status` shell variable, as shown in [Table 20](#).

Table 20: Checker Return Status

Value of <code>\$status</code>	Meaning
0	Everything is OK or the maximum severity of rule violations is NOTE.
1	There was an ERROR in the HDL analysis or the Checker exited incorrectly.
2	The Checker detected rule violations and the maximum severity of rule violations is WARNING.
3	The Checker detected rule violations and the maximum severity of rule violations is ERROR.
4	The Checker detected rule violations and the maximum severity of rule violations is FATAL.
5	Your Leda software license is invalid or not available.

Viewing Checker Results

For information on using the Checker GUI to view results captured in the leda.log file, see [“Post-processing Batch Mode Log Files” on page 128](#).

Checking the Environment

After you run the Leda Checker in batch mode using the `-full_log` switch, Leda creates a text file named `leda.inf` in the current working directory that captures information about the environment that Leda referenced for the check, including:

- Command-line used to invoke the Checker
- Settings for environment variables
- Configuration files used
- Policy versions used and full paths to their locations

You can use this file to make sure your results were based on the environment and configuration that you were expecting. For more information, see [“Post-processing Batch Mode Log Files” on page 128](#).

7

Using Leda GUI Mode

Introduction

This chapter explains what you can do using each of the menus and special features available from the Leda GUI, in the following major sections:

- [“Invoking the Checker/Specifier GUI” on page 170](#)
- [“Checking Your Environment” on page 171](#)
- [“Selecting a Text Editor” on page 172](#)
- [“The File Menu” on page 173](#)
- [“The Project Menu” on page 175](#)
- [“The Check Menu” on page 176](#)
- [“The Report Menu” on page 177](#)
- [“The View Menu” on page 178](#)
- [“The Window Menu” on page 178](#)
- [“The Help Menu” on page 178](#)
- [“Managing Source Files From the GUI” on page 181](#)
- [“Managing Library Units From the GUI” on page 184](#)

Invoking the Checker/Specifier GUI

First, set up your environment, as described in the section on “Configuring the Checker” in the *Leda Installation Guide*. Then, invoke the Checker as shown in the following example:

```
% $LEDA_PATH/bin/leda &
```

This brings up the Checker main window (see [Figure 46](#)). All the menus and functions in the Checker tool are also available from the Specifier tool. To invoke the Specifier GUI, use the following command:

```
% $LEDA_PATH/bin/leda -specifier &
```



Note

The Specifier and Checker GUIs are almost identical. The only difference is that the Specifier has a Policy Manager window (**Check > Configure**, then **Tool > Policy Manager**) that is not present in the Checker tool. You use the Policy Manager window to compile new rules for the Checker. You must have a Specifier license to run the Specifier tool.

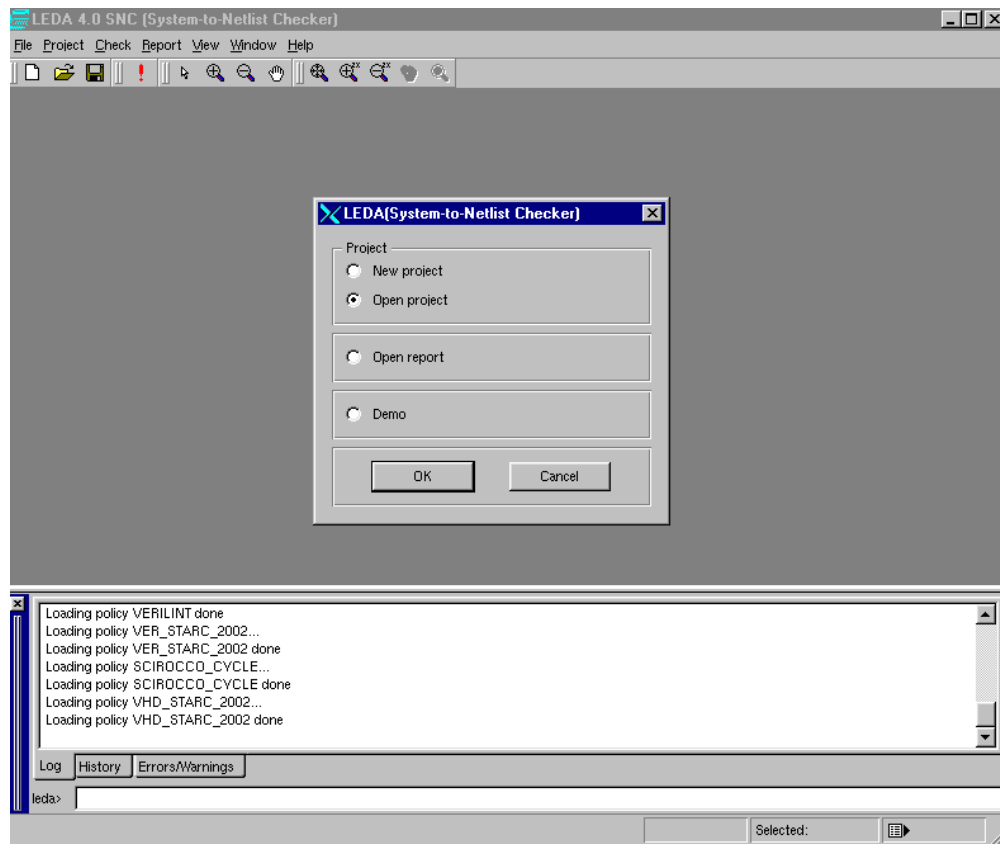


Figure 46: Leda Checker Main Window

Checking Your Environment

You can check the environment that Leda is referencing at any time after running the Leda Checker tool, either from the GUI or using the `-full_log` batch mode switch. This can be useful if you're not sure which configuration file you are using for a project, for example, because you can set up different projects to use different configurations for the prepackaged rules, as explained in [“Configuring the Rule Wizard” on page 73](#). You may also want to check the versions of the installed policies you are using, your settings for the various Leda configuration files, or the environment variables that Leda is currently using.

To check your environment, click on the Info Report tab on the right side of the main window. This displays the information that Leda is currently referencing, as shown in [Figure 47](#). If you don't see an Info Report on your display, this means that you are viewing a log file generated with a version of Leda prior to 3.0.

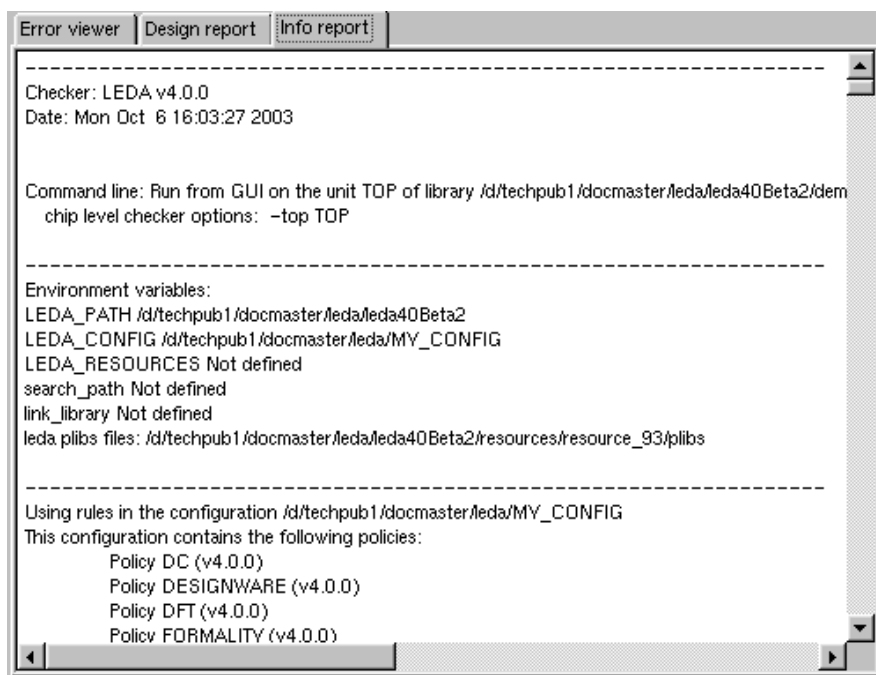


Figure 47: Leda Info Report Tab Display

For information on setting Leda environment variables, see [“Leda Environment Variables” on page 317](#).

When you run the command-line Checker on a project and use the `-full_log` switch, Leda stores the information you see in the Info Report tab in a text file typically named `leda.inf` in the same directory as the Checker log files, which is usually the current working directory. For more information on the Info Report, see [“Post-processing Batch Mode Log Files” on page 128](#).

Selecting a Text Editor

Leda comes with a default text editor that you can use to view and edit HDL source files in your Leda projects right from the Leda GUI. If you would rather use another popular text editor like Vi or XEmacs when you work with Leda, follow these steps:

1. Choose **File > Preferences > Source Settings** from the Specifier or Checker main window. This brings up the Application Preferences window. Choose Editor from the list on the left side of the window. The display changes to a window you can use to select a text editor other than the default Leda editor (see [Figure 48](#)).

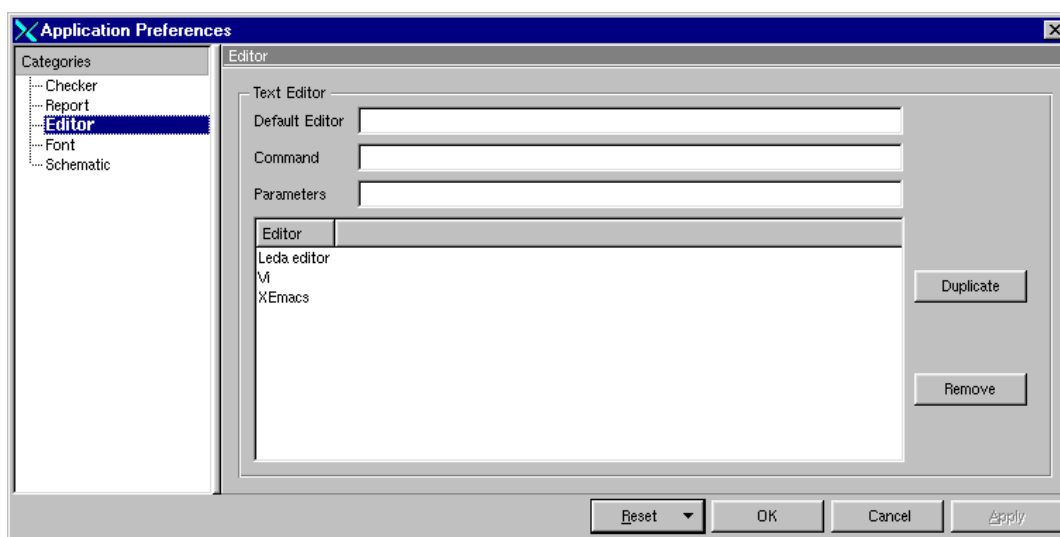


Figure 48: Set Text Editor Window

2. To select an editor, click Leda editor, Vi, or XEmacs in the selection pane. This changes the values displayed in the Default Editor, Command, and Parameters field at the top of the window to the editor you selected.
3. You can change the values in the Command and Parameters fields as needed. The defaults should work for most uses. If you want to change these values, first click on the editor you want in the selection pane, and then click the Duplicate button. This adds a copy of the selected editor to the selection pane.

4. With your copy of the editor selected, change the command and parameters as needed, just like you would in the UNIX shell. If your chosen editor is not in your \$path, make sure the Command field reflects the full path to the chosen executable (for example, /usr/local/bin/xemacs).
5. When you are done selecting your editor, click on the OK button. This saves your selection and dismisses the window. The next time you open an HDL source file from the GUI, Leda uses the editor you selected.

The File Menu

The File menu contains the choices described in [Table 21](#).

Table 21: File Menu Choices

Menu Item	Use
New	Brings up the New File dialog box, where you can specify a new file to open in the text editor. You can edit the file and recompile it if the file is a recognized HDL source file.
Open	Brings up the Open File dialog box, where you can specify an existing file to open in a text editor.
Recent Files	Displays files that you have used recently. Click on any of the displayed files to bring up a text editor on the file.
Preferences	Brings up the Application Preferences window, where you can set your preferences for: <ul style="list-style-type: none"> <u>Checker</u> Select the language (Verilog or VHDL), and checking mode (block-level, chip-level, netlist-checks). See “Setting & Saving Checker Preferences” on page 108. <u>Report</u> Select default sorting options for the Error Viewer. <u>Editor</u> Select vi, Xemacs, or the default Leda text editor. <u>Font</u> Select a font other than the default for the GUI display and reports. <u>Schematic</u> Select fill patterns and colors other than the defaults for the Path Viewer and Clock and Reset Tree Browsers.
Restore Last Session Preferences	Restores your Preferences to the way they were set the last time you used the GUI.

Table 21: File Menu Choices (Continued)

Menu Item	Use
Restore Default Preferences	Restores your Preferences to the default settings.
Save Preferences	Saves your preferences in a \$HOME/.synopsys_leda_prefs.tcl file that is used the next time you invoke the GUI (if the file exists).
AutoSave	Select this toggle switch if you want Leda to automatically save the Preferences you specify.
Quit	Exits the application.

The Project Menu

The Project menu contains the choices described in [Table 22](#).

Table 22: Project Menu Choices

Menu Item	Use
New	<p>Brings up the Project Creation Wizard window, which you can use to create a project that organizes your HDL source files. The Wizard takes you through the process window by window. Click the Next button on each window when you finish specifying your compiler options, libraries, and source files.</p> <p>After you completely specify your project, Leda creates a <i>project_name</i>-libs directory which contains a shell script named LVS_makelibs used to build HDL libraries, a Makefile to compile all source files into the correct library, and all libraries.</p>
Open	<p>Brings up the Open a Project window, which you can use to open an existing project. Navigate to the full path and name of an existing project, and click on the Open button. Leda locks this project automatically, and unlocks it again when you close it. However, if Leda exits abnormally while you have a project open, you get a message asking if the project should be unlocked.</p>
Edit	<p>Brings up the Project Update Wizard, which you can use to update an existing project by specifying new compiler options, libraries, or source files. The Wizard takes you through the process window by window. Click the Next button on each window when you finish making your changes.</p>
Save	<p>Saves the current project and all preferences that you specified. Leda prints save messages in the transcript window at the bottom of the display.</p>
Close	<p>Closes the current project. In most cases, it is best to save the current project before closing it.</p>
Delete	<p>Deletes the current project.</p>
Build	<p>VHDL only. Use this option when you have added a new VHDL source file or changed the hierarchy in the current VHDL project. Leda compiles all source files that contain design units in the current project's working libraries which you have modified or recompiled since the date of the last compilation.</p>
Build All	<p>VHDL only. Use this option when you have added a new VHDL source file or changed the hierarchy in the current VHDL project. Leda compiles all source files that contain design units in the current project's working libraries, regardless of when they were last modified or recompiled.</p>

Table 22: Project Menu Choices (Continued)

Menu Item	Use
Recent Projects	Displays a list of recently visited projects. Select the project you want to load from the list.

The Check Menu

The Check menu contains the choices described in [Table 23](#).

Table 23: Check Menu Choices

Menu Item	Use
Configure	Brings up the Leda Rule Wizard, which you can use to configure and select prepackaged and custom rules before checking your designs.
Load configuration	Brings up a pull-down menu where you can select a custom rule configuration or one of the four prebuilt rule configurations (see “Using Prebuilt Configurations” on page 99).
Run	Runs the Checker on the rules that you have selected, and prints messages in the transcript window at the bottom of the display. If necessary, presents a Get top module/design entity window, which you can use to specify the top-level unit in your design, and create test/reset clocks. Upon completion, the Checker displays results from the check in the Error Viewer on the right side of the main window.

The Report Menu

The Report menu contains the choices described in [Table 24](#).

Table 24: Report Menu Choices

Menu Item	Use
Open	Brings up the Select Log File window, which you can use to open a log file created by a previous run of the Checker, either in GUI or command-line mode. When the log file opens, the results for that check are displayed in the Error Viewer.
Save	Brings up the Save Report File As window, which you can use to save the results of the current Error Report in a log file for viewing later.
Save as ...	Brings up the Save Report File As window, which you can use to save the results of the current Error Report in a different location or with a different name for viewing later.
Save as HTML ...	Brings up the Save HTML Report File As window, which you can use to save the results of the current Error Report in HTML format and launch an HTML browser on the file.
Print	Prints the current Error Report.
Close	Closes the current Error Report.
Sort by ...	Brings up a menu that you can use to change the way your error messages are sorted in the Error Viewer. Available sorting options include Policy, Label, Severity, File, Module/unit, Language, and Master Rule.
Filter by ...	Brings up a menu that you can use to change the way your error messages are filtered in the Error Viewer. Available filtering options include Policy, Label, Severity, File, Module/unit, Language, and Master Rule. When you select one of these options, a window comes up that you can use to specify a regular expression for filtering your results for that item.
Summary	Toggles the Error Viewer summary report on and off.

The View Menu

The View menu contains tools that you can use to zoom in or pan around in the Path Viewer, and toggle the presence of the status window at the bottom of the display on or off. Turning this off gives you a little more real estate. Most of the functions in the View menu are also available using the toolbar icons.

The Window Menu

The Window menu contains functions that you can use to change the arrangement of the display windows in the Leda. You can tile the windows vertically or horizontally, or cascade them. You can also dock the active window on the top, bottom, right or left, or undock it.

The Help Menu

The Help menu provides access to the Leda documentation in PDF format (see [Table 25](#)).

Table 25: Help Menu Choices

Menu Item	Use
Leda Document Navigator	Brings up the Acrobat Reader on this PDF file, which provides an overview of the entire Leda documentation set, with hyperlinks to each manual. If you are not familiar with the Leda documentation, this is a good place to start.
Leda Installation Guide	Brings up the Acrobat Reader on this PDF file, which provides detailed instructions for installing and configuring the Leda software.
Leda Release Notes	Brings up the Acrobat Reader on this PDF file, which provides information about what's new and fixed bugs in the latest release of Leda.
Leda C Interface Guide	Brings up the Acrobat Reader on this PDF file, which documents the C API for writing custom netlist checker rules in C or C++.
Leda Tcl Interface Guide	Brings up the Acrobat Reader on this PDF file, which documents the Tcl API for writing custom netlist checker rules in Tcl.
Leda Rule Specifier Tutorial	Brings up the Acrobat Reader on this PDF file, which provides a hands-on tutorial for how to write new rules for Leda using the VRSL and VerSL rule specification languages.

Table 25: Help Menu Choices (Continued)

Menu Item	Use
<i>VRSL Reference Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information on VRSL, which you use to write new rules for VHDL designs.
<i>VeRSL Reference Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information on VeRSL, which you use to write new rules for Verilog designs.
<i>Leda User Guide</i>	Brings up the Acrobat Reader on this PDF file (this manual), which provides comprehensive procedures for how to use Leda.
<i>Leda General Coding Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules in the Leda policy.
<i>Leda RMM Coding Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules in the RMM policy.
<i>Leda IEEE Verilog Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules in the IEEE Verilog policy.
<i>Leda IEEE VHDL Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules that come in the IEEE VHDL policy.
<i>Leda Design Compiler Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules that come in the Design Compiler policy.
<i>Leda VCS Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules that come in the VCS policy.
<i>Leda Scirocco Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules that come in the Scirocco policy.
<i>Leda DesignWare Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules that come in the DesignWare policy.
<i>Leda Formality Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules that come in the Formality policy.

Table 25: Help Menu Choices (Continued)

Menu Item	Use
<i>Leda DFT Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules that come in the DFT policy.
<i>Leda Verilint Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules that come in the Verilint policy.
<i>Leda STARC DSG Verilog Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules that come in the STARC DSG Verilog policy.
<i>Leda STARC DSG VHDL Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules that come in the STARC DSG VHDL policy.
<i>Leda Design Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules that come in the Design policy.
<i>Leda Constraints Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules that come in the Constraints policy.
<i>Leda Xilinx Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules that come in the Xilinx policy.
<i>Leda Power Rules Guide</i>	Brings up the Acrobat Reader on this PDF file, which provides detailed reference information for the prepackaged rules that come in the Power policy.
Leda on the Web	Launches a browser session and takes you to the Leda product page on the Synopsys Web site.
About Leda	Brings up a splash page that displays the version of Leda that you are using.

Managing Source Files From the GUI

The Files tab shown in [Figure 49](#) appears on the left side of the main window.

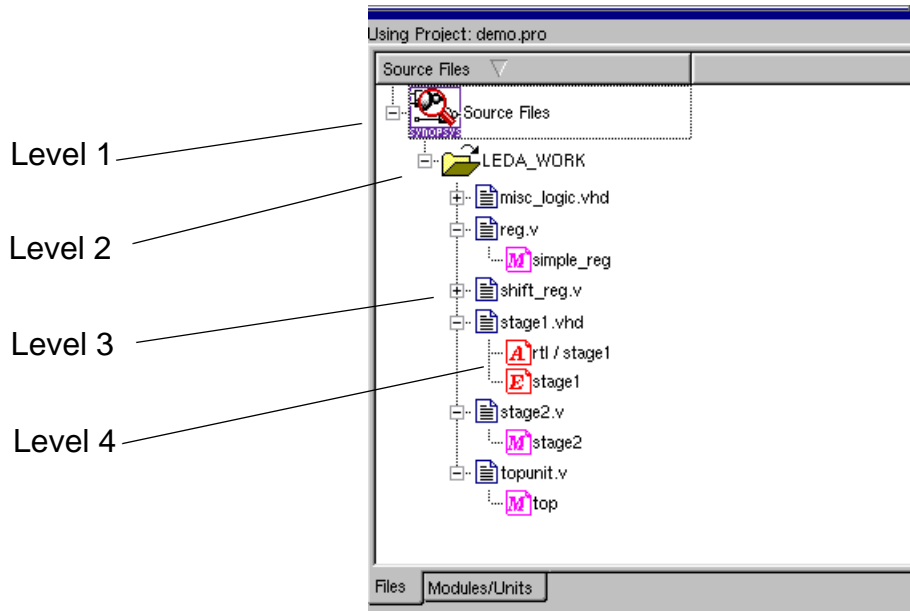


Figure 49: Source File Manager Window

The Source File Manager represents the HDL source files loaded for your project in a graphical tree with four levels, as explained in [Table 26](#).

Table 26: Source File Levels in Display

Level	Description
Level 1	Shows the root level of the current project.
Level 2	Shows the working libraries of the current project.
Level 3	Shows the relative names of the source files in each working library.
Level 4	Shows the design units present in the source files.

You can expand or contract the hierarchical display by clicking on the (+) or (-) box icons. Note that each level in the hierarchy has a different icon associated with it. Right click on any file in the display and choose “Edit the file” from the popup menu to open a text editor on the file.

Using Pop-up Menus in the Files Tab

When you right-click on icons at different levels of the hierarchy in the Files tab from the main window, special pop-up menus appear that are geared for the different levels in the hierarchy, as explained in the following subsections, from top to bottom in the project source file hierarchy:

- “Project Pop-up Menu” on page 182
- “Library Pop-up Menu” on page 182
- “Source File Pop-up Menu” on page 183
- “Unit Pop-up Menu” on page 183

Project Pop-up Menu

To activate the Project pop-up menu, hold down the right mouse button on the Source Files icon or label. This makes the choices shown in [Table 27](#) available.

Table 27: Project Pop-up Menu Choices

Menu Item	Use
Add a library ...	Adds a selected library to the project as a working library.
Build the project	Compiles all project source files in their order of appearance.
Save the project	Saves the current project’s working libraries, resource libraries, and source files.
Close all libraries	Closes all opened levels except the project. You can do the same thing by double-clicking on the project icon or name.

Library Pop-up Menu

To activate the Library pop-up menu, hold down the right mouse button on a Library icon or label. This makes the choices shown in [Table 28](#) available.

Table 28: Library Pop-up Menu Choices

Menu Item	Use
Add files ...	Brings up the Add Files window, which you can use to add source files to the corresponding library.
Build the library	Compiles all source files in the library in their order of appearance.
Remove the library from the project	Removes the selected library from the current project.

Table 28: Library Pop-up Menu Choices (Continued)

Menu Item	Use
Remove the library from the disk	Removes the selected library from the current project and from the disk.

Source File Pop-up Menu

To activate the Source File pop-up menu, hold down the right mouse button on a source file icon or name. This makes the choices shown in [Table 29](#) available.

Table 29: Source File Pop-up Menu

Menu Item	Use
Edit the file	Brings up a text editor on the selected HDL source file.
Compile the file	Compiles the selected source file into the corresponding library.
Remove the file from the project	Removes the selected source file from the project.
Remove the file from the disk	Removes the selected source file from the corresponding library and from the disk.

Unit Pop-up Menu

To activate the Unit pop-up menu, hold down the right mouse button on a unit or name. This makes the choices shown in [Table 30](#) available.

Table 30: Unit Pop-up Menu

Menu Item	Use
Edit the unit	Edits the source file of the selected design unit at the beginning of its description.
Compile the file	Compiles the source file containing the selected design unit into the corresponding library.

Managing Library Units From the GUI

The Modules/Units tab shown in [Figure 50](#) appears in the main window when you select that tab.

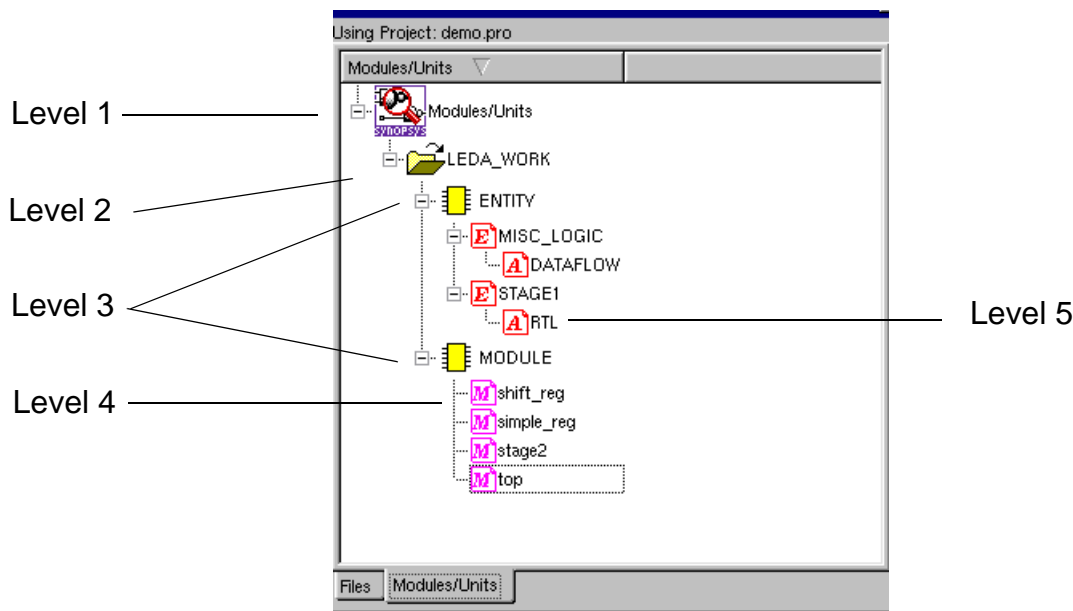


Figure 50: Library Unit Manager Window

The Library Unit Manager represents the HDL source files loaded for your project in a graphical tree with five levels, as explained in [Table 31](#).

Table 31: Library Unit Levels in Display

Level	Description
Level 1	Shows the current project, at the root of the hierarchy.
Level 2	Shows the working libraries of the current project in upper-case lettering.
Level 3	Shows the primary kinds of units in each working library,
Level 4	Shows the primary units in each working library. Leda determines this by referencing the corresponding library.
Level 5	Shows the secondary units in each working library. These units directly depend on their primary units.

You can expand or contract the hierarchical display by clicking on the (+) or (-) box icons. Note that each level in the hierarchy has a different icon associated with it. Right click on any file in the display and choose “Edit the file” from the pop-up menu to open a text editor on the file.

Using Pop-up Menus in the Modules/Units Tab

When you right-click on icons at different levels of the hierarchy in the Modules/Units tab from the main window, special pop-up menus appear that are geared for the different levels in the hierarchy, as explained in the following subsections, from top to bottom in the library unit hierarchy:

- “Project Pop-up Menu” on page 185
- “Library Pop-up Menu” on page 186
- “Unit Pop-up Menu” on page 186

Project Pop-up Menu

To activate the Project pop-up menu, hold down the right mouse button on a Modules/Units icon or label. This makes the choices shown in [Table 32](#) available.

Table 32: Project Pop-up Menu Choices

Menu Item	Use
Execute Checkers	Executes the Checker on all design units in the selected library.
Show Errors on <i>project_name</i> for >	Brings up another pop-up menu that lists the names of prepackaged policies. You can use these choices to filter the results in the Error Viewer accordingly.
Refresh View	Refreshes the view.
Close all Units	Collapses the hierarchical display so that units are not shown.

Library Pop-up Menu

To activate the Library pop-up menu, hold down the right mouse button on a library icon or label. This makes the choices shown in [Table 33](#) available.

Table 33: Library Pop-up Menu Choices

Menu Item	Use
Add files ...	Brings up a window that you can use to add files to the project.
Build the library	Makes Leda compile all the files in the selected library.
Remove the library from the project	Removes the selected library from the current project.
Remove the library from the disk	Removes the selected library from the current project and from the disk.

Unit Pop-up Menu

To activate the Unit pop-up menu, hold down the right mouse button on a unit icon or label. This brings up a pop-up menu that you can use to Edit the file.

Generating Log Files in GUI Mode

When you create a project in the GUI mode, a directory leda-logs is created in the present working directory. Files leda.log and leda.inf are created in this directory.

8

Using Leda Tcl Shell Mode

Introduction

This chapter provides reference information for the built-in Tcl commands implemented in Leda. You can use these Tcl commands to configure rules, manage projects, and control your Leda Checker runs (see [“Built-in Tcl Commands” on page 189](#)).

But first, here’s an overview of how to get started in the Tcl shell in the following sections:

- [“Invoking Leda in Tcl shell Mode” on page 187](#)
- [“Enabling Netlist Checks” on page 188](#)
- [“Changing Leda Modes” on page 188](#)
- [“Sourcing a Tcl Script in Leda” on page 188](#)
- [“Collections” on page 190](#)
- [“Regular Expressions” on page 192](#)

Invoking Leda in Tcl shell Mode

To invoke Leda in Tcl shell mode:

```
% leda +tcl_shell [-project project_name]  
% leda +tcl_shell batch_command_line_args/options [-project project_name]
```

The Leda Tcl commands work from the Tcl console at the bottom of the GUI window or from the Tcl prompt in the shell when you are not using the GUI. In both cases, the Tcl prompt looks like this:

```
leda>
```

In Tcl shell mode, Leda saves the Tcl commands from your setup and configuration files in a `leda_command.log` file in the current working directory. This file is overwritten for each new session.

Enabling Netlist Checks

In addition to the built-in Tcl commands documented in this chapter, you have access to an extensive set of Tcl Design Query Language (DQL) functions that you can use to interactively query your elaborated design database in Tcl shell mode. These are the same functions that you can use to develop your own custom netlist-checking rules. For complete reference information on the Tcl DQL API, see the [Leda Tcl Interface Guide](#).

To enable the DQL with an existing project:

```
% leda +tcl_shell
leda> project_open existing_project_name
leda> elaborate
```

If you don't want to open an existing project, you can also enable the DQL by reading in some HDL source files, specifying the top-level unit, and elaborating the design as follows:

```
% leda +tcl_shell (to start the tool)
leda> read_verilog netlist.v (or a set of files)
leda> current_design name_of_top_level_unit
leda> elaborate
```

Changing Leda Modes

For information on how to switch back and forth between GUI mode, Tcl shell mode, and batch mode to perform different tasks with Leda, see [“Using Leda in Batch, GUI, and Tcl Shell Modes” on page 30](#).

Sourcing a Tcl Script in Leda

You can write a Tcl script that uses the commands documented in this chapter and source them in the Leda environment using the `+tcl_file` option:

```
% leda +tcl_file script.tcl
```

Built-in Tcl Commands

The built-in Tcl commands provide alternative ways to access tool functionality that is also available using the Leda GUI and the batch mode Checker. The built-in Tcl commands fall into three major categories:

- Rule commands that you can use to manage the rules that you use to check your HDL design (see [“Rule Tcl Command Reference” on page 197](#)).
- Project commands that you can use to manage your Leda projects (see [“Project Tcl Command Reference” on page 253](#)).
- Checker commands that you can use to control your runs with the Leda Checker (see [“Checker Tcl Command Reference” on page 271](#)).

Getting Help on Leda Tcl Commands

You can get help on any of the commands documented in this section using the `help -v` option from the Tcl prompt, as follows:

```
leda> help -v leda_tcl_command
```

For example, to get help on the `rule_manage_policy` command:

```
leda> help -v rule_manage_policy
rule_manage_policy    # Create a policy
    -policy <policy_name>  (Give the policy name)
    [-format <language_name>]
                        (Set the language name:
                        Values: verilog, vhdl)
    [-ruleset <name>]      (Give the ruleset name)
    [-templateset <name>] (Give the templateset name)
    command               (Execute the command:
                        Values: create, delete, compile)
    [file_s]              (List of rule files to be compiled)
```

Collections

A collection is a group of objects exported to Tcl user interface. A set of commands to create and to manipulate collections is provided as an integral part of the user interface. The collection commands are divided into two categories:

- Commands that create collections of objects for use by another command,
- Commands that query objects for your viewing.

The result of a command that creates a collection is a Tcl object that can be passed along to another command. For a query command, although the visible output looks like a list of objects (a list of object names is displayed), the result of a query command is an empty string. An empty string "" is equivalent to an empty collection, which is a collection with zero elements.

Leda collections can contain the following objects:

- Cell
- Port
- Pin
- Net
- Power_domain

Each object is defined by its type (cell, port, pin, net or power_domain) and its name (string displayed to the user interface).

The following table describes the attributes of each of the objects that Leda supports:

Table 34: Attributes of Objects supported by Collection

Object	Attribute	Type	Description
Cell	base_name	string	The leaf name of a cell. For example, the base_name of cell U1.U2.U3 is U3.
	full_name	string	The complete name of a cell. For example, the full name of cell U3 within cell U2 within cell U1 is U1.U2.U3. The full_name attribute is not affected by current_instance.
	object_class	string	The class of the object. This is a constant equal to cell.

Table 34: Attributes of Objects supported by Collection

Object	Attribute	Type	Description
Net	base_name	string	The leaf name of a net. For example, the base name of net i1.i1z1 is i1z1. You cannot set this attribute.
	full_name	string	The complete name of a net. For example, the full_name of net i1z1 within cell i1 is i1.i1z1. The full_name attribute is not affected by current instance. The full_name attribute is read-only.
	object_class	string	The class of the object. This is a constant, equal to net.
Pin	base_name	string	The leaf name of a pin. For example, the base name of pin U1.U2.Z is Z. You cannot set this attribute.
	full_name	string	The complete name of a pin to the top of the hierarchy. For example, the full name of pin Z on cell U2 within cell U1 is U1.U2.Z. The setting of the current instance has no effect on the full name of a pin.
	object_class	string	The class of the object. This is a constant, equal to pin.
Port	base_name	string	The leaf name of a port. For example, the base name of port i1.i1z1 is i1z1. You cannot set this attribute.
	full_name	string	The complete name of a port to the top of the hierarchy. For example, the full name of port Z on cell U2 within cell U1 is U1.U2.Z. The setting of the current instance has no effect on the full name of a port.
	object_class	string	The class of the object. This is a constant, equal to port.
Power_domain	base_name	string	The name of a power domain.
	full_name	string	It is same as the base name.
	object_class	string	The class of the object. This is a constant, equal to power_domain.

Leda provides Tcl commands to create and manipulate collections. For more information, see [“Rule Tcl Command Reference” on page 197](#).

Current Limitation

Leda collections cannot contain the following objects:

- Lib
- Design

The following table list the commands that are not supported by Leda, but when executed by Leda will not result in an error or warning.

Table 35:

all_connected	Used to create a collection of objects connected to another.
all_fanin	Used to create a collection of pins/ports or cells in the fanin of specified sinks.
all_fanout	Used to create a collection of pins/ports or cells in the fanout of specified sources.
get_designs	Used to create a collection of designs.
get_lib_cells	Used to create a collection of library cells.
get_lib_pins	Used to create a collection of library cell pins.
get_libs	Used to create a collection of libraries.
get_timing_paths	Used to create a collection of timing paths.

Regular Expressions

You can use simple wildcard characters like * and ? and also complete regular expressions with the collection.

For example:

```
leda> set gc [get_cells -regexp {i(1|2)_.*}]
```


The `-regexp` option tells Leda to view the patterns argument as a regular expression. In addition, the pattern matching operators in the filter expression (`=~` and `!~`) also use regular expressions. The following commands, illustrates the basic regular expressions (both commands do the same thing):

```
leda> get_cells blk* -filter "base_name =~ AN*"
leda> get_cells -regexp {blk.*} -filter "base_name =~ AN"
```

You need to use the `-nocase` option with the `-regexp` option to perform a case-insensitive search in Leda.

Using Regular Expressions with Hierarchy

Using the `-regexp` option along with the `-hierarchical` option is different from using a wildcard pattern with the `-hierarchical` option. The hierarchical searches with regular expressions are always matched with the full name of the objects. The behavior of hierarchical searches with regular expressions are as follows:

- Using `-regexp` option alone matches the leaf names in the current instance. For example:

```
leda> get_cells -regexp i1*.
```

- Using `-regexp` option with `-hierarchical` option matches full names, relative to the current instance, for each object found at or below the current instance. This is independent of the existence of hierarchy separators in the pattern. For example, to create a collection of `i1.i2.n1`, `i1.i21.n1`, `i1.i2.i3.n1`, etc., the command should be as follows:

```
leda> get_cells -regexp i1/i2.* /n1
```

- Using `-regexp` does not provide a direct method to match leaf names at each level of the hierarchy. However, you can emulate a method of matching leaf names by using filters. For example, you can use `-regexp` to do the same as the following command:

```
leda> get_cells n1 -hierarchical
```

To do this, use the `base_name` attribute, that is the leaf name of the cell. For example:

```
leda> get_cells -regexp -hierarchical ".*" -filter {base_name == n1}
```

Anchoring Regular Expressions

By default, Leda automatically anchors regular expression patterns. For example, the pattern `blk.*` is considered the same as `^blk.*$`, that is usually the intended behavior.

If you want a less restricted matching style, prefix and suffix the pattern with `.*` to unanchor the pattern (that is, not do an exact match). For example, to get any cells that contain U1:

```
leda> get_cells -regexp {.*U1.*}
```

The above command matches U1, U11, U1A, U1_23, plus ZU1, ZZU1, hello_U1, etc.,

Using Regular Expressions with Busses

You should be careful when using a regular expression to match buses, because the bus characters “(” and “)” are part of the command language. In addition, the usage varies slightly depending on whether the command argument is a string or a list.

For a string command argument, the following example shows the correct form. The expression argument to `filter_collection` is a string.

```
leda> filter_collection -regexp [get_ports *] {full_name =~ a\([0-1]\)}
```

The above regular expression matches ports a(0) and a(1). A single backslash (\) must precede the bracket.

For a list command argument, the syntax depends on how you specify the list. Consider the following example, which uses the `get_ports` command. The “patterns” argument to `get_ports` is a list.

```
leda> get_ports -regexp [list {a\([0-1]\)}]
```

```
leda> get_ports -regexp {{a\([0-1]\)}}
```

These two commands are equivalent. Proper list forms require single backslash quoting “\”, just like string arguments. It is recommended that you use a properly formatted list for a list argument, especially in this situation. However, when you pass a single string into the “patterns” argument, double backslash quoting “\\” is required. For example:

```
leda> get_ports -regexp {a\\([0-1]\\)}
```

The double backslash is required because the promotion of the string to a list consumes one of the backslashes.

Filter Expressions

You can filter collections by using the `-filter` option with the primary commands that create collections. You can also use the `filter_collection` command.

Using the -filter Option

Many commands that create collections accept a `-filter` option that specifies a filter expression. A filter expression is a string composed of a series of logical expressions describing a set of constraints you want to place on a collection.

Each sub expression of a filter expression is a relation contrasting an attribute name (such as `area` or `direction`) with a value (such as `43` or `input`), by means of an operator (such as `==` or `!=`).

The following command gets the cells in U1 that have an area no greater than 12 or reference a design (or library cell) named AN2P, AO2P, etc. The command then assigns the collection to the cells variable as follows:

```
leda> set Cells [get_cells "*" -filter {full_name =~ U* || \
                                     base_name !~ "A*P"}]
```

The filter language supports the following logical operators:

- AND or `&&` - Logical AND (case insensitive)
- OR or `||` - Logical OR (case insensitive)

To enforce the evaluation order, you need to group the logical expressions with parentheses. Otherwise, Leda evaluates expression from left to right.

The filter language supports the following relational operators:

- Equal (`==`)
- Not Equal (`!=`)
- Greater than (`>`)
- Less than (`<`)
- Greater than or equal to (`>=`)
- Less than or equal to (`=<`)
- Matches pattern (`~=`)
- Does not matches pattern (`!~`)

In the following filter expression,

```
{full_name =~ U* || base_name !~ "A*P"}
```

`full_name` is an attribute (or identifier), the operator `~=` is a relational operator, `U*` is a value, and the operator `||` is the logical OR operator.

The filter language also supports the following existence operators:

- `defined`

- undefined

An existence operator determines if an attribute is defined for an object or not. For example:

```
sense == defined(sense)
```

The right side of a relation can consist of a string or a numeric literal. You do not need to enclose strings in quotation marks. This method is useful because a filter expression is usually the value for an argument, and the entire expression is enclosed in quotation marks.

The following command illustrates that, you need not enclose the word in with quotation marks.

```
leda> set port [get_ports * -filter {full_name =~ A/B/*}]
```

However, if an expression contains characters that are part of the filter language syntax, you must use curly braces to enclose the expression and double quotation marks to enclose string operands. Since parentheses are part of the filter language, they are double quoted in the following example and the complete expression is grouped in curly braces:

```
leda > set x [filter_collection $ports {base_name =~ "D(*)"}]
```

Rule Tcl Command Reference

Following is command reference information for the built-in Tcl commands that you can use to manage the rules that run against your HDL design files. To see the help for all `rule_*` commands implemented in Leda, use the `help -v` switch from the Tcl prompt in the Tcl console at the bottom of the GUI or in the Tcl shell when you are not running the GUI:

```
leda> help -v rule_*
```



Attention

Options shaded in grey color are ignored by Leda .

is_64bit

Use the `is_64bit` command to check if the operating system that you are currently working is 32/64bit.

Syntax

```
is_64bit
```

This command returns 1 if the operating system is 64-bit.

add_to_collection

Use the `add_to_collection` command to add objects to a collection. The result is a new collection.

Syntax

```
add_to_collection [-unique] collection1 object_spec
```

Arguments

<code>-unique</code>	Removes duplicates from the result.
<code>collection1</code>	Base collection.
<code>object_spec</code>	Objects to add.

all_clocks

Use the `all_clocks` command to create a collection of all clocks of a design.

Syntax

```
all_clocks
```

all_inputs

Use the all_inputs command to create a collection of all input ports of a design.

Syntax

```
all_inputs
```

all_instances

Use the all_instances command to create a collection of all instances of a design.

Syntax

```
all_instances
```

all_outputs

Use the all_outputs command to create a collection of all output ports of a design.

Syntax

```
all_outputs
```

all_registers

Use the all_registers command to create a collection of all register cells or pins.output ports of a design.

Syntax

```
all_registers [-no_hierarchy]
```

Arguments

-no_hierarchy Limits only to the current level of hierarchy

append_to_collection

Use the append_to_collection command to add objects to a collection. The result modifies the collection variable.

Syntax

```
append_to_collection [-unique] var_name object_spec
```

Arguments

-unique	Removes duplicates from the result.
var_name	Variable that holds the collection.
object_spec	Objects to append.

create_operating_conditions

Use the `create_operating_conditions` command to create the volatile operating conditions and associate it to a library.

Syntax

```
create_operating_conditions [-name name] \  
-library { lib_name1 lib_name2...}
```

Arguments

-name	Specify the operating condition name.
-library	Specify the library names.

compare_collections

Use the `compare_collections` command to compare two collections and see if they contain the same objects. It returns 0 if they contain the same objects.

Syntax

```
compare_collections [-order_dependent] collection1 collection2
```

Arguments

-order_dependent	Considers the of objects.
collection1	Base collection.
object_spec	The collection to compare with the with the base collection.

connect_power_domain

Use the `connect_power_domain` command to connect a power domain to power net information. Syntax

```
connect_power_domain [-primary_power_net name] \
  [-primary_ground_net name] [-backup_power_net net] \
  [-backup_ground_net name] [-internal_power_net name] \
  [-internal_ground_net name] power_domain_name
```

Arguments

<code>-primary_power_net</code>	Specify the primary power net name.
<code>-primary_ground_net</code>	Specify the primary ground net name.
<code>-backup_power_net</code>	Specify the power net name.
<code>-backup_ground_net</code>	Specify the ground net name.
<code>-internal_power_net</code>	Specify the power net name.
<code>-internal_ground_net</code>	Specify the ground net name.

copy_collections

Use the `copy_collections` command to duplicate the contents of a collection, resulting in a new collection.

Syntax

```
copy_collections collection1
```

Arguments

<code>collection1</code>	Collection to copy.
--------------------------	---------------------

create_power_domain

Use the `create_power_domain` command to create a power domain.

Syntax

```
create_power_domain <domain_name> [-power_down]
  [-power_down_ack <net or pin>] [-power_down_ctrl <net or pin>]
  [-object_list <cell set>]
```

Arguments

<code>domain_name</code>	Specify the power domain name within a quoted string.
<code>-power_down</code>	Specify this option to power down.

<code>-power_down_ctrl</code>	Specify the single bit net that powers down the domain. If the value of the net is 1, then the domain is powered-down (always active high). If this option is not used, then the corresponding power domain is always on.
<code>-power_down_ack</code>	Specify the single bit net that acknowledges the power down state of a domain.
<code>cell set</code>	Specify the list of cells that should be associated with this power domain. If no cell set is present, then this command creates the top level power domain for the design.

create_power_net_info

Use the `create_power_net_info` command to create a power net information.

Syntax

```
create_power_net_info name [-power] [-gnd] [-voltage_range {min max}] \
  [-voltage_values {val1 ... valN}] [-source_port design_port]
```

Arguments

<code>name</code>	Specify the name of the power net.
<code>-voltage_range</code>	Specify the legal voltage range for this power net.
<code>-voltage_values</code>	Specify the legal voltage values for this power net.
<code>-source_port</code>	Specify the top-level port in the design that is the source of this power net.

delete_operating_conditions

Use the `delete_operating_conditions` command to delete the operating conditions.

Syntax

```
delete_operating_conditions [-name name] \
  -library { lib_name1 lib_name2...}
```

Arguments

<code>-name</code>	Specify the operating condition name.
<code>-library</code>	Specify the library names.

disable_isolation_cell_recognition

Use the `disable_isolation_cell_recognition` command to disable the recognition of isolation cells.

Syntax

```
disable_isolation_cell_recognition
```

enable_isolation_cell_recognition

Use the `enable_isolation_cell_recognition` command to force the checker to accept any standard cell having the AND or the OR function as a possible isolation cell. In such a case, the criteria for a standard cell to be recognized as an isolation cell for a given power domain is as follows:

- Either one of the inputs of the cell is directly or indirectly (through combinatorial logic) connected to an output of the given power domain, or the output of the cell is directly/indirectly connected to an input of the given power domain.
- An input of the cell is directly/indirectly connected to the control signal(s) specified for the given power domain.

Syntax

```
enable_isolation_cell_recognition [-strict]
```

Arguments

<code>-strict</code>	Enables the strict matching mode when recognizing an isolation cell.
----------------------	--

filter_collection

Use the `filter_collection` command to filter a collection, resulting in a new collection.

Syntax

```
filter_collection [-regex] [-nocase] collection1 expression
```

Arguments

<code>-regex</code>	Operators <code>=~</code> and <code>!~</code> use regular expressions.
<code>-nocase</code>	Regular expression matches are case sensitive. Use this option to make it case insensitive.
<code>collection1</code>	Collection to filter.
<code>expression</code>	Filter expression

foreach_in_collection

Use the `foreach_in_collection` command to iterate over the elements of a collection.

Syntax

```
foreach_in_collection itr_var collections body
```

Arguments

<code>itr_var</code>	Specifies the name of the iterator variable.
<code>collections</code>	Specifies a list of collections over which to iterate.
<code>body</code>	Specifies a script to execute per iteration.

get_all_input_boundaries_from_power_domain

Use the `get_all_input_boundaries_from_power_domain` command to get the list of input pins of cells used by the checks on power domains.

Syntax

```
get_all_input_boundaries_from_power_domain <inferred_power_domain_name>
```

get_all_output_boundaries_from_power_domain

Use the `get_all_output_boundaries_from_power_domain` command to get the list of output pins of cells used by the checks on power domains.

Syntax

```
get_all_output_boundaries_from_power_domain <inferred_power_domain_name>
```

get_cells

Use the `get_cells` command to create a list of cells.

Syntax

```
get_cells [-hierarchical] [-filter expression] [-quiet] [-regexp]
          [-nocase] [-exact] [-of_objects objects] [patterns]
```

Arguments

<code>-hierarchical</code>	Specify this option to find objects throughout hierarchy.
<code>expression</code>	Specify the expression to filter collection with this expression.
<code>-quiet</code>	Use this option to suppress all messages.
<code>-regexp</code>	Patterns are regular expressions.

-nocase	Regular expression matches are case sensitive. Use this option to make it case insensitive.
-exact	Wildcards are treated as plain characters.
-of_objects	Specify this option to get cells related to these objects.
patterns	Specify the list of cell name patterns.

get_clocks

Use the `get_clocks` command to create a collection of clocks.

Syntax

```
get_cells [-hierarchical] [-filter expression] [-quiet] [-regexp]
          [-nocase] [-exact] [patterns]
```

Arguments

-hierarchical	Specify this option to find objects throughout hierarchy.
<i>expression</i>	Specify the expression to filter collection with this expression.
-quiet	Use this option to suppress all messages.
-regexp	Patterns are regular expressions.
-nocase	Regular expression matches are case sensitive. Use this option to make it case insensitive.
-exact	Wildcards are treated as plain characters.
-of_objects	Specify this option to get cells related to these objects.

get_nets

Use the `get_nets` command to create a list of pins.

Syntax

```
get_nets [-hierarchical] [-filter expression] [-quiet] [-regexp]
          [-nocase] [-exact] [-of_objects objects] [patterns]
```

Arguments

-hierarchical	Specify this option to find objects throughout hierarchy.
<i>expression</i>	Specify the expression to filter collection with this expression.
-quiet	Use this option to suppress all messages.
-regexp	Patterns are regular expressions.

-nocase	Regular expression matches are case sensitive. Use this option to make it case insensitive.
-exact	Wildcards are treated as plain characters.
-of_objects	Specify this option to get ports related to these objects.
patterns	Specify the list of net name patterns.

get_nth_power_net

Use the `get_nth_power_net` command to return the name of the `nth` power net.

Syntax

```
get_nth_power_net name
```

Arguments

`name` Specify the power domain name.

get_object_name

Use the `get_object_name` command to get the full name of the object in a single-object collection.

Syntax

```
get_object_name collection
```

Arguments

`collection` Specifies the name of the collection that contains the single object whose name is requested.

get_power_cells

Use the `get_power_cells` command to return the cells of a given power domain.

Syntax

```
get_power_cells name
```

Arguments

`name` Specify the power domain name.

get_power_down

Use the `get_power_down` command to return the power down net associated with the given power domain.

Syntax

```
get_power_down name
```

Arguments

name Specify the power domain name.

get_power_down_ack

Use the `get_power_down_ack` command to return the power down ack net associated with the given power domain.

Syntax

```
get_power_down_ack name
```

Arguments

name Specify the power domain name.

get_power_net_max_voltage

Use the `get_power_net_max_voltage` command to return the maximum value of the power net voltage values.

Syntax

```
get_power_net_max_voltage name
```

Arguments

name Specify the power net name.

get_power_net_min_voltage

Use the `get_power_net_min_voltage` command to return the minimum value of the power net voltage values.

Syntax

```
get_power_net_min_voltage name
```

Arguments

name Specify the power net name.

get_power_net_source_port

Use the `get_power_net_source_port` command to return the design port that is the source of the power net.

Syntax

```
get_power_net_source_port name
```

Arguments

`name` Specify the power net name.

get_power_net_type

Use the `get_power_net_type` command to return the type of the power net (GND or POWER).

Syntax

```
get_power_net_type name
```

Arguments

`name` Specify the power net name.

getn_power_net

Use the `getn_power_net` command to return the number of power nets.

Syntax

```
getn_power_net
```

get_pins

Use the `get_pins` command to create a list of nets.

Syntax

```
get_pins [-hierarchical] [-filter expression] [-quiet] [-regexp]  
[-nocase] [-exact] [-of_objects objects] [patterns]
```

Arguments

`-hierarchical` Specify this option to find objects throughout hierarchy.
`expression` Specify the expression to filter collection with this expression.
`-quiet` Use this option to suppress all messages.
`-regexp` Patterns are regular expressions.

-nocase	Regular expression matches are case sensitive. Use this option to make it case insensitive.
-exact	Wildcards are treated as plain characters.
-of_objects	Specify this option to get pins related to these objects.
patterns	Specify the list of pin name patterns.

get_ports

Use the `get_ports` command to create a list of ports.

Syntax

```
get_ports [-hierarchical] [-filter expression] [-quiet] [-regexp]
          [-nocase] [-exact] [-of_objects objects] [patterns]
```

Arguments

-hierarchical	Specify this option to find objects throughout hierarchy.
<i>expression</i>	Specify the expression to filter collection with this expression.
-quiet	Use this option to suppress all messages.
-regexp	Patterns are regular expressions.
-nocase	Regular expression matches are case sensitive. Use this option to make it case insensitive.
-exact	Wildcards are treated as plain characters.
-of_objects	Specify this option to get ports related to these objects.
<i>patterns</i>	Specify the list of port name patterns.

get_power_domains

Use the `get_power_domains` command to create a list of power domains.

Syntax

```
get_power_domains [-filter expression] [-quiet] [-regexp]
                  [-nocase] [-exact] [patterns]
```

Arguments

<i>expression</i>	Specify the expression to filter collection with this expression.
-quiet	Use this option to suppress all messages.
-regexp	Patterns are regular expressions.

-nocase	Regular expression matches are case sensitive. Use this option to make it case insensitive.
-exact	Wildcards are treated as plain characters.
patterns	Specify the list of port name patterns.

infer_power_domain

Use the `infer_power_domain` command to infer a power domain in a PG-Netlist from a power net.

Syntax

```
infer_power_domain [-power_net <name>] domain_name ]
```

Arguments

-power_net	Specify the power net name.
domain_name	Specify the power domain name.

infer_power_domains

Use the `infer_power_domains` command to infer a power domain from the RTL (`$power`).

Syntax

```
infer_power_domains [-verbose]
```

Arguments

-verbose	Specifies that it is in verbose mode
----------	--------------------------------------

index_collection

Use the `index_collection` command to extract an object from a collection. Given a collection and an index it, if the index is in range, this command extracts the object at that index and creates a new collection containing only that object. The base collection remains unchanged.

Syntax

```
index_collection collection1 index
```

Arguments

collection1	Specifies the collection to be searched.
-------------	--

`index` Specifies the index into the collection. Allowed values are integers from 0 to `sizeof_collection - 1`.

print_config_summary

Use the `print_config_summary` command to print the configuration summary on the console. The summary is displayed after the rules checking is done and is saved to `$PWD/leda_config.log`.

Syntax

```
print_config_summary
```

query_objects

Use the `query_objects` command to search for and display the objects in the database.

Syntax

```
query_objects [-verbose] [-class class_name] [-truncate elem_count]
              object_spec
```

Arguments

<code>-verbose</code>	Displays the class of each object found. By default, only the name of each object is listed. With this option, each object name is preceded by its class, as in "cell:U1/U3"
<code>-class class_name</code>	Establishes the class for a named element in the <code>object_spec</code> . Valid classes are <code>design</code> , <code>cell</code> , <code>net</code> , and so on.
<code>-truncate elem_count</code>	Truncates display to <code>elem_count</code> elements. By default, up to 100 elements are displayed. To see more or less elements, use this option. To see all elements, set <code>elem_count</code> to 0.
<code>object_spec</code>	Provides a list of objects to find and display. Each element in the list is either a collection or an object name. Object names are explicitly searched for in the database with class <code>class_name</code> .

remove_from_collection

Use the `remove_from_collection` command to remove objects from a collection, resulting in a new collection. The base collection remains unchanged.

Syntax

```
remove_from_collection base_collection object_spec
```

Arguments

<code>base_collection</code>	Specifies the base collection to be copied to the result collection. Objects matching <code>object_spec</code> are removed from the result collection.
<code>object_spec</code>	Specifies the objects to be removed.

remove_isolation_cell

Use the `remove_isolation_cell` command to specify the isolation cell to be removed from the list of isolation cells created by consecutive calls to the `set_isolation_cell` command

Syntax

```
remove_isolation_cell {list of cell names} | -instance {instance_list}
```

Only the isolation cells defined with `set_isolation_cell` can be removed (not the DB cells).

For more information, see the [Leda Power Rules Guide](#).

remove_level_shifter

Use the `remove_level_shifter` command to specify the level shifter cells to be removed from the list of level shifter cells created by consecutive calls to the `set_level_shifter` command.

Syntax

```
remove_level_shifter {list of cell names}
```

Only the cells defined with `set_level_shifter` can be removed (not the DB cells).

For more information, see the [Leda Power Rules Guide](#).

remove_power_domain

Use the `remove_power_domain` command to remove a power domain from the design.

Syntax

```
remove_power_domain [ -all ] | object_list
```

Arguments

- all Remove all the power domains from the design.
- object_list Specify the list of power domains to be removed.

For more information, see the [Leda Power Rules Guide](#).

remove_power_net_info

Use the `remove_power_net_info` command to remove a power net specification.

Syntax

```
remove_power_net_info [ -all ] | domain_name
```

Arguments

- all Remove all the power net from the design.
- domain_name Specify the name of the power net.

For more information, see the [Leda Power Rules Guide](#).

report_clock_gating_cells

Use the `report_clock_gating_cells` command to list all defined clock gating cells.

Syntax

```
report_clock_gating_cells
```

Example

This command will list all the defined clock gating cells.

```
leda> report_clock_gating_cells
```

For more information, see the [Leda Power Rules Guide](#).

report_enable_pin

Use the `report_enable_pin` command to list the enable pin if any for a given cell.

Syntax

```
report_clock_gating_cells cell_name
```

Example

This command will list the enable pin of the cell IC12V.

```
leda> report_enable_pin IC12V
Enable pin of IC12V: EN
leda> report_enable_pin IC12VB
Enable pin of IC12V: <not found>
```

For more information, see the [Leda Power Rules Guide](#).

report_isolation_cells

Use the `report_isolation_cells` command to list all the defined isolation cells.

Syntax

```
report_isolation_cells
```

Example

This command will list all the isolation cells.

```
leda> report_isolation_cells
```



Warning

This command will report automatically recognized isolation cells (when `enable_isolation_cell_recognition` is set) only after one of the rules `ICINSALL`, `ICINSIN` or `ICINSOUT` check has been executed.

For more information, see the [Leda Power Rules Guide](#).

report_level_shifter

Use the `report_level_shifter` command to list all the defined level shifters.

Syntax

```
report_level_shifter
```

Example

This command will list all the defined level shifters.

```
leda> report_level_shifter
```

For more information, see the [Leda Power Rules Guide](#).

report_operating_conditions

Use the `report_operating_conditions` command to report all or specific operating conditions of a given library.

Syntax

```
report_operating_conditions [-name name] \  
-library { lib_name1 lib_name2...}
```

Arguments

- | | |
|----------|---------------------------------------|
| -name | Specify the operating condition name. |
| -library | Specify the library names. |

report_pin_voltages

Use the `report_pin_voltage` command to list all pin voltage values defined for a given cell.

Syntax

```
report_pin_voltages cell_name
```

Example

This command will list all the defined pin voltages of cell LS9_12V.

```
leda> report_pin_voltages LS9_12V
```

For more information, see the [Leda Power Rules Guide](#).

report_power_domain

Use the `report_power_domain` command to report the information about the power domains.

Syntax

```
report_power_domain object_list
```

Arguments

object_list Specify the list of power domains to be reported.

Example

This command will list all the defined power domains.

```
leda> report_power_domain
```

For more information, see the [Leda Power Rules Guide](#).

report_power_net_info

Use the report_power_net_info command to remove the power net specifications.

Syntax

```
report_power_net_info [object_list ]
```

Arguments

object_list Specify the list of cells.

For more information, see the [Leda Power Rules Guide](#).

report_power_pins

Use the report_power_pins command to report the power pins of the given cell.

Syntax

```
report_power_pins cell
```

Arguments

cell Specify the cell name.

For more information, see the [Leda Power Rules Guide](#).

report_power_switches

Use the report_power_switches command to report the power switches.

Syntax

```
report_power_switches
```

reset_isolation_cell_recognition

Use the `reset_isolation_cell_recognition` command to reset the isolation cell recognition database.

Syntax

```
reset_isolation_cell_recognition
```

For more information, see the [Leda Power Rules Guide](#).

rule_deselect

Use the `rule_deselect` command to specify a rule that you want to deselect for checking. You can put `rule_deselect` commands in a configuration file in your configuration directory that Leda reads automatically (see “[Deactivating Rules with a Rule Configuration File](#)” on page 102) or enter them interactively at the Tcl prompt in the GUI. If you do not specify any options with `rule_deselect`, Leda deselects all rules in your configuration by default.

Syntax

```
rule_deselect [-rule label] [-ruleset ruleset_name] \  
  [-policy policy_name] [-all] [-vhdl] [-verilog] [-file file_name] \  
  [-section {begin_line end_line}] [-through name] [-instance name]
```

Arguments

<code>-rule</code>	Specify the <i>label</i> of the rule you want to deselect for checking.
<code>-ruleset</code>	Specify the name of the <i>ruleset_name</i> you want to deselect for checking.
<code>-policy</code>	Specify the name of the <i>policy_name</i> you want to deselect for checking.
<code>-all</code>	Deselect all rules for checking.
<code>-vhdl</code>	Deselect all VHDL rules for checking. It works only for block-level rules.
<code>-verilog</code>	Deselect all Verilog rules for checking. It works only for block-level rules.
<code>-file</code>	Specify the <i>file_name</i> where you want the rule deselected.
<code>-section</code>	Specify the <i>begin_line</i> and <i>end_line</i> in the <code>-file file_name</code> (see above) where you want the rule deselected.

- through** For chip-level and netlist rules, specify the instance *name* through which to deselect the rule for checking. This deactivates the rule for checking if the rule's tracing information passes through the specified instance *name*, including errors completely contained in the specified instance *name*.
- instance** For chip-level and netlist rules, specify the instance *name* in which to deselect the rule for checking. This deactivates the rule for checking only if the rule's tracing information is completely contained in the specified instance *name*.

Example

This command does not return a value when it completes successfully. The following example deselects rule B_1000 just for Verilog:

```
leda> rule_deselect -rule B_1000 -verilog
```

This next example does the same thing, except that it deselects the rule for checking on both VHDL and Verilog project source files:

```
leda> rule_deselect -rule B_1000
```

And this last example deselects all rules in the Formality policy:

```
leda> rule_deselect -policy FORMALITY
```

rule_get_parameter

Use the `rule_get_parameter` command to get a list of valid parameters for the specified rule. Not all rules have configurable parameters. If you execute this command on a rule that does not have configurable parameter, the tool returns you to the prompt without displaying any information.

Syntax

```
rule_get_parameter rule_label
```

Arguments

`rule_label` Specify the rule label.

Example

The following example shows that rule B_4200 (Entity name should end in `_ENT`) has a parameter called `ENTITY_NAME` that you can configure using a regular expression to match something other than `_ENT`:

```
leda> rule_get_parameter B_4200
{ENTITY_NAME {_ENT$} REGEXP}
```

rule_get_selection

Use the `rule_get_selection` command to find out how many rules are selected for checking based on the argument you provide. This command returns a number that tells you how many rules are selected at that level in the hierarchy. For example, if you run `rule_get_selection` on the top-level descriptor (*language*) and specify Verilog, the command returns the number of Verilog rules selected for checking (1062 in the Demo project that comes with the Checker). If you run this command on an individual rule, you could get a return value of 2 if the rule is selected for checking in VHDL and Verilog, 1 if it is selected for checking in just one language, or 0 if it is not selected for checking at all. To check to see if an individual rule is selected for checking regardless of language, use the `-fast` switch. When you use `-fast`, Leda returns a 1 if the rule is selected for checking or a 0 if it is not.

Syntax

```
rule_get_selection [language.][policy.][ruleset.]]rule [-fast] [-total]
```

Arguments

<code>rule</code>	Specify the <i>language</i> , <i>policy</i> , <i>ruleset</i> , and <i>rule</i> , a subset of these, or just the <i>rule</i> .
<code>-fast</code>	Use the <code>-fast</code> switch to find out if the specified rule is selected for checking. When you use <code>-fast</code> , Leda returns a 1 (true) if the rule is selected for checking or a 0 (false) if it is not.
<code>-total</code>	Use the <code>-total</code> switch to see the total number of rules selected for checking and the total rules available in the current configuration, based on the arguments you provide.

Example

The following example returns the selection status for rule DFT_019 from the Design For Test (DFT) policy:

```
leda> rule_get_selection DFT_019
2
```

In this example Leda returns a 2, which tells you that this rule is selected for checking in VHDL and Verilog, which could be interesting to note for mixed-language designs.

In this next example for the same rule, we use the `-fast` switch, so Leda reports true (1), because this rule is selected for checking:

```
leda> rule_get_selection DFT_019 -fast
1
```

rule_get_all_masters_from_topic

The `rule_get_all_masters_from_topic` command returns a list of information for a topic, including master IDs. Sometimes, prepackaged rules from different policies have duplicate functionality because they cover the same topic. In such cases, the similar rules share a common master ID.

Syntax

```
rule_get_all_masters_from_topic topic_name
```

Arguments

`topic_name` Specify the topic for which you want to obtain the master IDs. To find out the legal topic names, first use the [propagate](#) command.

Returned Values

Returns a list of master information for the specified topic, including Master IDs, which are used to identify redundant rules that appear in different policies for commonly checked items.

Example

The following example returns the master information for all rules pertaining to clocks.

```
leda> rule_get_all_masters_from_topic clocks
{M_0423 {Avoid gated clock in the design}} {M_0306 {Avoid using
asynchronous logic}} {M_0276 {Avoid using both positive-edge and
negative-edge triggered flip-flops in your design}} {M_0295 {Buffers
should not be explicitly added to clock path}} {M_0418 {Clocks must not
be used as data}} {M_0419 {Data must be registered by 2 flipflops when
changing clock domain}} {M_0327 {Do not use event definitions for
clocks}} {M_0417 {Information on the number of clock signals in the
design}} {M_0326 {Internally generated clock detected (block level)}}
{M_0420 {Internally generated clock detected (chip level)}} {M_0983
{Multi-bit expression (e.g a[2:0]) used as clock}} {M_0325 {Multi-bit
expression used as clock}} {M_0323 {Multiple clocks in this unit
detected}} {M_0322 {Multiple event control statement in a task}} {M_0711
{Multiplexed clock is detected}} {M_0321 {Nested event control in a
task}} {M_0421 {No gated clock except in clock generator CKGEN}} {M_0452
{Only one clock is allowed in an always block}} {M_0805 {Register with
fixed value clock is detected}} {M_0422 {Use rising edge clock in the
design}} {M_0324 {Use rising edge clock in this unit}} {M_0451 {Use
rising edge flipflop.}}
```

rule_get_all_rules_from_master_id

After you identify a master ID that it is interesting using the [propagate](#) command, you can then find all individual rules that share that master ID using `rule_get_all_rules_from_master_id` command.

Syntax

```
rule_get_all_rules_from_master_id master_id
```

Arguments

`master_id` Specify the master ID.

Returned Values

Returns a list of rules for the specified master ID in *policy ruleset rule* format.

Example

To find all rules governed by the M_0423 masterID, which concerns avoiding gated clocks in the design, use the following command. Note that there are four rules that check for this problem; they appear in the DesignWare, Leda General Coding Guidelines, RMM, and Scirocco_Cycle policies.

```
leda> rule_get_all_rules_from_master_id M_0423
{DESIGNWARE ARCHITECTURE A_5C_R_B} {LEDA CLOCKS C_1207}
{RMM_RTL_CODING_GUIDELINES CLOCKS_AND_RESETS G_543_1} {SCIROCCO_CYCLE
CHIP_LEVEL SC_300}
```

rule_get_all_topics

Use the `rule_get_all_topics` command to get a list of legal rule topic names. Use this command in conjunction with the [propagate](#) command.

Syntax

```
rule_get_all_topics
```

Arguments

None.

Returned Values

Returns a list of all topics in the current configuration.

Example

The following example returns all topics covered by rules that are selected in the default configuration for the Demo project that comes with the tool:

```
leda> rule_get_all_topics
CLOCKS CODING_FOR_SYNTHESIS CODING_STYLE DATA_TYPES DESIGN_STRUCTURE DFT
EXPRESSIONS HDL_LAYOUT HDL_NAMING MODELING RESETS RTL_NAMING
SIMULATION_CYCLE_MODE SIMULATION_MISMATCH SIMULATION_PERFORMANCE
STATEMENTS STATE_MACHINES
```

rule_get_configuration

Use the `rule_get_configuration` command to return the configuration for the specified rule, where configuration means the rule label, message, severity, HTML-based help file name, and master ID.

Syntax

```
rule_get_configuration -policy policy_name -ruleset ruleset_name \  
-rule rule_name [-type attribute_name] [-format language]
```

Arguments

- policy Specify the policy name for the specified rule.
- ruleset Specify the ruleset name for the specified rule.
- rule Specify the rule name.
- type Get the specified configuration information. Legal values for *attribute_name* include message, severity, html, masterid, selection, and deselection_in_file. Default is all configuration information.
- format Get the HDL languages that this rule applies to.

Returned Values

Returns the configuration information for the specified rule.

Example

The following example returns the message text for rule SC_301 from the Scirocco_Cycle policy:

```
leda> rule_get_configuration -policy scirocco_cycle \  
-ruleset chip_level -rule sc_301 -type message  
In any cycle mode partitioned block, the clock should be an input to the  
block.
```

rule_get_current_configuration

Use this command to get the current configuration name.

Syntax

```
rule_get_current_configuration [-writable]
```

Arguments

-writable Get the directory where any changes to the configuration will be saved (if you have write permissions).

Returned Values

Returns the full path to the directory that contains your current configuration.

Example

The following example shows that the current configuration is Leda-optimized, which is one of the prebuilt configurations that comes with Leda (see [“Using Prebuilt Configurations” on page 99](#)):

```
leda> rule_get_current_configuration
/d/techpub1/docmaster/leda/leda403_software/configurations/
Leda-optimized
```

rule_get_policies

Use the `rule_get_policies` command to get a list of available policies in the current configuration for a given language (vhdl or verilog).

Syntax

```
rule_get_policies [-format language]
```

Arguments

`-format` Set the *language*. Legal values are verilog and vhdl. Default is all.

Returned Values

Returns a list of policies for the specified language available in the current configuration.

Example

The following example returns a list of available policies for Verilog in the current configuration:

```
leda> rule_get_policies -format verilog
DC DESIGN DESIGNWARE DFT FORMALITY IEEE_RTL_SYNTH_SUBSET LEDA
RMM_RTL_CODING_GUIDELINES VCS VERILINT VER_STARC_DSG
```


rule_get_policy_attributes

Use the `rule_get_policy_attributes` command to get a list of attributes for the specified language, policy, and attribute name.

Syntax

```
rule_get_policy_attributes -policy policy_name [-format language] \  
[-name attribute_name]
```

Arguments

- policy Specify the policy name. Legal values include LEDA, DESIGNWARE, SCIROCCO, FORMALITY, DC, VCS, VERILINT, IEEE_VERILOG, IEEE_VHDL, VER_STARC_DSG, and VHD_STARC_DSG. To get a list of legal policy names, use the [propagate](#) command.
- format Specify the language. Legal values include verilog and vhdl. Default is all.
- name Get information for the specified *attribute_name*. Legal values include version, icon, write_permissions, language, and path. Default is all.

Returned Values

Returns the value of an attribute for the specified policy and language.

Example

The following example returns the version for the Leda General Coding Guidelines policy:

```
leda> rule_get_policy_attributes -policy LEDA -format verilog \  
-name version  
4.0.3
```

rule_get_predefined_configurations

Use the `rule_get_predefined_configurations` command to get a list of the predefined configurations that are set up for Leda. A predefined configuration contains a set of rules from multiple policies for different HDL checking needs. The current list of predefined configurations includes RTL, Gate-level, Leda-optimized, and Leda-classic (see [“Using Prebuilt Configurations” on page 99](#)).

Syntax

```
rule_get_predefined_configurations
```

Arguments

None.

Example

```
leda> rule_get_predefined_configurations  
Gate-level Leda-classic Leda-optimized RTL
```

rule_get_rules

Use the `rule_get_rules` command to get a list of available rules for a given policy, ruleset, and language.

Syntax

```
rule_get_rules -policy policy_name -ruleset ruleset_name \  
  [-format language]
```

Arguments

- policy** Specify the *policy_name*. Legal values include LEDA, DESIGNWARE, SCIROCCO, FORMALITY, DC, VCS, VERILINT, IEEE_VERILOG, IEEE_VHDL, VER_STARC_DSG, and VHD_STARC_DSG. To get an updated list of legal policy names, use the [propagate](#) command.
- ruleset** Specify the *ruleset_name*. To get a list of legal ruleset names, first use the [propagate](#) command.
- format** Set the *language*. Legal values include verilog and vhdl. Default is all.

Returned Values

Returns a list of rules for the specified policy, ruleset, and language.

Example

The following example returns a list of Verilog rules in the DESIGN_STRUCTURE ruleset of the Leda General Coding Guidelines policy:

```
leda> rule_get_rules -policy LEDA -ruleset DESIGN_STRUCTURE \  
  -format verilog  
B_1000 B_1001 B_1005 B_1006 B_1010 B_1011 B_1013 C_1000 C_1001 C_1002  
C_1003 C_1004 C_1005 C_1006 C_1007 C_1008 C_1009
```

rule_get_ruleset_attributes

Use the `rule_get_ruleset_attributes` command to return the values for the specified ruleset attributes.

Syntax

```
rule_get_ruleset_attributes -policy policy_name -ruleset ruleset_name \  
  [-format language] [-name attribute_name]
```

Arguments

- policy** Specify the *policy_name*. Legal values include LEDA, DESIGNWARE, SCIROCCO, FORMALITY, DC, VCS, VERILINT, IEEE_VERILOG, IEEE_VHDL, VER_STARC_DSG, and VHD_STARC_DSG. To get an updated list of legal policy names, use the [propagate](#) command.
- ruleset** Specify the *ruleset_name*. To get a list of legal ruleset names, use the [propagate](#) command.
- format** Get the language. Legal values include verilog and vhdl. Default is all.
- name** Get information about the specified attribute name. Legal values include language, icon, chip, file, and obsolete. Default is all.

Returned Values

Returns the attribute values for the specified ruleset.

Example

The following example returns a list of all attribute values for the DESIGN_STRUCTURE ruleset of the Leda General Coding Guidelines policy:

```
leda> rule_get_ruleset_attributes -policy LEDA -ruleset \  
  DESIGN_STRUCTURE -format verilog  
{language {VERILOG VHDL}} {icon {warning.bmp warning.msk yellow}}  
{chip 1} {file {/d/techpub1/docmaster/leda/leda4023R_software/  
.leda_config/rules/leda/./LEDA.sl 10 /d/techpub1/docmaster/leda/  
leda4023R_software/.leda_config/rules/leda/./LEDA.rl 10}}  
{obsolete {{} 0 0}}
```

rule_get_rulesets

Use the `rule_get_rulesets` command to get the rulesets for a given policy.

Syntax

```
rule_get_rulesets -policy policy_name [-format language]
```

Arguments

- `-policy` Specify the policy name. Legal values include LEDA, DESIGNWARE, SCIROCCO, FORMALITY, DC, VCS, VERILINT, IEEE_VERILOG, IEEE_VHDL, VER_STARC_DSG, and VHD_STARC_DSG. To get an updated list of legal policy names, use the [propagate](#) command.
- `-format` Set the language. Legal values include verilog and vhdl. Default is all.

Returned Values

Returns the ruleset names for the specified policy.

Example

The following example returns a list of all rulesets in the Leda General Coding Guidelines policy:

```
leda> rule_get_rulesets -policy LEDA
DATA_TYPES CLOCKS DESIGN_STRUCTURE EXPRESSIONS HDL_NAMING RESETS
RTL_NAMING RTL_SYNTHESIS STATEMENTS STATE_MACHINES SYSTEMVERILOG
HDL_LAYOUT
```

rule_get_templateset_attributes

Use the `rule_get_templateset_attributes` command to get the attributes for a given templateset.

Syntax

```
rule_get_templateset_attributes -policy policy_name \  
-templateset templateset_name [-name attribute_name]
```

Arguments

- policy Specify the *policy_name*. Legal values include LEDA, DESIGNWARE, SCIROCCO, FORMALITY, DC, VCS, VERILINT, IEEE_VERILOG, IEEE_VHDL, VER_STARC_DSG, and VHD_STARC_DSG. To get an updated list of legal policy names, use the [propagate](#) command.
- templateset Specify the *templateset_name*. Legal values depend on the specified policy. To find out the templatesets used in a given policy, use the [propagate](#) command.
- name Set the *attribute_name*. Legal values include language, icon, chip, file, and obsolete. Default is all.

Returned Values

Returns the attribute names for the specified policy and templateset.

Example

The following example returns all attribute names for the CLOCK_EDGES templateset used in the Leda General Coding Guidelines policy.

```
leda> rule_get_templateset_attributes -policy LEDA \  
-templateset CLOCK_EDGES  
{language VHDL} {file {/d/techpub1/docmaster/leda/leda403_software/  
.leda_config/rules/leda/./templateset/edges.rl 24}} {obsolete {{} 0}}
```

rule_get_templatesets

Use the `rule_get_templatesets` command to get the names of all templatesets used by a given policy. A templateset is like a Verilog module. It contains a set of template declarations. No commands are allowed in templatesets, but they can contain other templateset units. Rulesets can contain template declarations, commands, and other templateset units.

Syntax

```
rule_get_templatesets -policy policy_name [-format language]
```

Arguments

- policy Specify the policy name. Legal values include LEDA, DESIGNWARE, SCIROCCO, FORMALITY, DC, VCS, VERILINT, IEEE_VERILOG, IEEE_VHDL, VER_STARC_DSG, and VHD_STARC_DSG. To get an updated list of legal policy names, use the [propagate](#) command.
- format Set the *language*. Legal values include verilog and vhdl. Default is all.

Returned Values

Returns the names of all templatesets used in the specified policy.

Example

The following example returns the names of all templatesets used in the Leda General Coding Guidelines policy:

```
leda> rule_get_templatesets -policy LEDA
CLOCK_EDGES IEEE_DECLARATIONS STD_DECLARATIONS RMM_CLOCK_EDGES
RMM_PROCESSES
```

rule_link

Use the `rule_link` command to add a netlist checker custom rule developed in C/C++ to the list of rules that run the next time the Checker is executed.

Syntax

```
rule_link file.ext rule_label
```

Arguments

file.ext Specify the object or shared library file that contains the compiled rule source code, where *ext* is platform-dependent:

- Solaris—*file.o* (object file)
- Linux—*file.so* (shared library file)
- HP-UX—*file.sl* (shared library file)

rule_label Specify the C function name.

Example

The following example links the C object file `toto.o` to `rule_1`.

```
leda> rule_link toto.o rule_1
```

rule_load

Use the `rule_load` command to load all available policies.

Syntax

```
rule_load [-fast]
```

Arguments

`-fast` Do not open policy libraries.

Example

```
leda> rule_load
```


rule_load_configuration

Use the `rule_load_configuration` command to load a configuration. A configuration specifies the set of rules that you want to check.

Syntax

```
rule_load_configuration [-check] [directory_name]
```

Arguments

<code>-check</code>	Save only if the configuration has been modified.
<code>directory_name</code>	Specify the full path to the directory that contains the configuration file. If you don't specify a <code>directory_name</code> , Leda loads the default configuration.

Example

The following example loads the configuration file located in the specified directory. This configuration now determines which rules are checked the next time you run the Checker.

```
leda> rule_load_configuration /u/me/MyLedaConfig
```

rule_manage_policy

Use the `rule_manage_policy` command to create, compile, or delete a policy, ruleset, or templateset.

Syntax

```
rule_manage_policy -policy policy_name [-ruleset ruleset_name] \  
  [-templateset templateset_name] [-format language] \  
  command [files list_of_files]
```

Arguments

- policy Specify the *policy_name* to manage. Legal values for prepackaged policies include LEDA, DESIGNWARE, SCIROCCO, FORMALITY, DC, VCS, VERILINT, IEEE_VERILOG, IEEE_VHDL, VER_STARC_DSG, and VHD_STARC_DSG. To get an updated list of legal policy names, use the [propagate](#) command. You can also specify a name for a new policy that you want to create.
- ruleset Set the *ruleset_name* to manage. Default is all.
- templateset Set the *templateset_name* to manage. Default is all.
- format Set the language. Legal values include verilog and vhdl.
- command* Specify the action you want to take on a policy, ruleset, or templateset. Legal values for *command* include create, compile, and delete.
- files* Specify the list of files to create or compile. Use only with the create and compile commands.

Example

The following example compiles two .sl VerSL source code files into a new policy for Verilog called MY_POLICY:

```
leda> rule_manage_policy -policy MY_POLICY -format verilog \  
  compile -files first.sl second.sl
```

rule_patch

Use the `rule_patch` command to replace the object or shared library file for a prepackaged rule developed in C/C++ with an updated version. This is a handy way to patch in revised rules without having to rebuild policies.

Syntax

```
rule_patch file.ext rule_label
```

Arguments

file.ext Specify the object or shared library file that contains the revised compiled rule source code, where *ext* is platform-dependent:

- Solaris—*file.o* (object file)
- Linux—*file.so* (shared library file)
- HP-UX—*file.sl* (shared library file)

rule_label Specify the C function name.

Example

The following example replaces the C object file `toto.o` for `rule_1`.

```
leda> rule_patch toto.o rule_1
```

rule_save_configuration

Use the `rule_save_configuration` command to save the current configuration. When you execute this command without arguments, Leda saves the current configuration in the same directory where it was found (usually `$LEDA_CONFIG`).

Syntax

```
rule_save_configuration [-check] [dir_name]
```

Arguments

`-check` Save the configuration only if it has changed.

dir_name Specify a different directory where you want to save the current configuration.

Example

The following example saves the current configuration in the specified directory:

```
leda> rule_save_configuration /u/me/LEDA_CONFIG
The current configuration is saved into /u/me/LEDA_CONFIG
```

rule_get_current_configuration

Use the `rule_get_current_configuration` command to get the current configuration name.

Syntax

```
rule_get_current_configuration [-writable]
```

Arguments

`-writable` Get the directory where any changes to the configuration will be saved (if you have write permissions).

Returned Values

Returns the full path to the directory that contains your current configuration.

Example

The following example shows that the full path to the current configuration; in this case Leda-optimized, which is one of the prebuilt configurations that come with Leda (see [“Using Prebuilt Configurations” on page 99](#)).

```
leda> rule_get_current_configuration
/d/techpub1/docmaster/leda/leda403_software/configurations/
Leda-optimized
```

rule_set_default_configuration

Use the `rule_set_default_configuration` command to select the recommended set of rules for any policy (set of prepackaged rules).

Syntax

```
rule_set_default_configuration -policy policy_name [-check]
```

Arguments

- policy Specify the *policy_name*. Legal values include LEDA, DESIGNWARE, SCIROCCO, FORMALITY, DC, VCS, VERILINT, IEEE_VERILOG, IEEE_VHDL, VER_STARC_DSG, and VHD_STARC_DSG. To get an updated list of legal policy names, use the [propagate](#) command.
- check Check to see if the specified policy has a default configuration.

Returned Values

When used with the `-check` switch, this command returns true (1) if the specified policy is selected for checking or false (0) if the specified policy is not selected for checking. When used without the `-check` switch, this command returns a message indicating that the recommended set of rules for the specified policy was loaded.

Example

The following example selects the recommended set of rules for the Leda policy, which contains 300 prepackaged rules. When you set the recommended set of rules for the Leda policy, there are 206 recommended rules selected for checking.

```
leda> rule_set_default_configuration -policy LEDA
Setting default configuration for the policy LEDA done.
```

rule_set_predefined_configuration

Use this command to load one of the predefined configurations that are set up for Leda. A predefined configuration contains a set of rules from multiple policies for different HDL checking needs. The current list of predefined configurations includes RTL, Gate-level, Leda-optimized, and Leda-classic (see [“Using Prebuilt Configurations” on page 99](#)).

Syntax

```
rule_set_predefined_configuration config_name
```

Arguments

config_name	Set the <i>config_name</i> . For a list of legal configuration names, use the propagate command.
-------------	--

Example

The following example loads the Leda-optimized prebuilt configuration and shows the policies that are loaded as a result. Note that Leda-optimized must be typed exactly as shown because the *config_name* argument is case-sensitive.

```
leda> rule_set_predefined_configuration Leda-optimized
Loading policy DC...
Loading policy DC done
Loading policy DESIGNWARE...
Loading policy DESIGNWARE done
Loading policy DESIGN...
Loading policy DESIGN done
Loading policy DFT...
Loading policy DFT done
Loading policy FORMALITY...
Loading policy FORMALITY done
Loading policy IEEE_RTL_SYNTH_SUBSET...
Loading policy IEEE_RTL_SYNTH_SUBSET done
Loading policy LEDA...
Loading policy LEDA done
Loading policy RMM_RTL_CODING_GUIDELINES...
Loading policy RMM_RTL_CODING_GUIDELINES done
Loading policy VCS...
Loading policy VCS done
Loading policy VERILINT...
Loading policy VERILINT done
Loading policy VER_STARC_DSG...
Loading policy VER_STARC_DSG done
Loading policy SCIROCCO_CYCLE...
Loading policy SCIROCCO_CYCLE done
Loading policy VHD_STARC_DSG...
Loading policy VHD_STARC_DSG done
```

rule_select

Use the `rule_select` command to select a rule, ruleset, or policy for checking that was previously deselected (see “[rule_deselect](#)” on page 216). If you use this command on a rule, ruleset, or policy that was not previously deselected, it has no effect. A common idiom that you can use to make sure this command works as expected is to write a configuration file that first deselects all rules and then selects just the ones you want to check your design with. For example, your configuration file can be as simple as this if you want to check only the rules in the Design policy (netlist checks):

```
rule_deselect -all
rule_select -p DESIGN
```

If you do not specify any options with `rule_select`, Leda selects all rules in your configuration by default. You can put `rule_select` commands in a configuration file in your configuration directory that Leda reads automatically or enter them interactively at the Tcl prompt.

Syntax

```
rule_select [-rule label] [-ruleset ruleset_name] \  
  [-policy policy_name] [-all] [-vhdl] [-verilog] [-file file_name] \  
  [-section {begin_line end_line}] [-through name] [-instance name]
```

Arguments

<code>-rule</code>	Specify the <i>label</i> of the rule you want to select for checking.
<code>-ruleset</code>	Specify the <i>ruleset_name</i> you want to select for checking.
<code>-policy</code>	Specify the <i>policy_name</i> you want to select for checking.
<code>-all</code>	Select all rules for checking. This is the default.
<code>-vhdl</code>	Select just VHDL rules for checking. It works only for block-level rules.
<code>-verilog</code>	Select just Verilog rules for checking. It works only for block-level rules.
<code>-file</code>	Specify the <i>file_name</i> where you want the rule selected.
<code>-section</code>	Specify the <i>begin_line</i> and <i>end_line</i> in the <code>-file file_name</code> (see above) where you want the rule selected.
<code>-through</code>	For chip-level and netlist rules, specify the instance <i>name</i> through which to select the rule for checking. This activates the rule for checking if the rule’s tracing information passes through the specified instance <i>name</i> , including errors completely contained in the specified instance <i>name</i> .

-instance For chip-level and netlist rules, specify the instance *name* in which to select the rule for checking. This activates the rule for checking only if the rule's tracing information is completely contained in the specified instance *name*.

Example

The following example selects rule B_1000 from the Leda General Coding Guidelines policy on both VHDL and Verilog project source files. This command does not return a value if it succeeds, but you can confirm that the rule has been selected on your next run with the tool or look at the tail of the configuration file for the currently loaded configuration, where you will see your `rule_select` command saved.

```
leda> rule_select -rule B_1000
```

rule_set_html

Use the `rule_set_html` command to set the name of the HTML help file for a given rule. This command is best used for custom rules that you develop, because Leda's prepackaged rules already have HTML help files specified for them.

Syntax

```
rule_set_html -rule [language.][policy.][ruleset.]]rule \  
-html html_filename
```

Arguments

-rule Specify the rule label. Optionally specify the *language* for rules that apply to both VHDL and Verilog. Also, optionally specify the *policy* and *ruleset*.

-html Specify the *html_filename* that contains help information for that rule.

Example

The following example sets the HTML help file name for RULE_1 to the RULE_1.html file. This command does not return a value when it completes successfully.

```
leda> rule_set_html -rule RULE_1 -html /u/me/leda/RuleHelp/RULE_1.html
```


rule_set_message

Use the `rule_set_message` command to set the message text for a given rule.

Syntax

```
rule_set_message -rule [language.][policy.][ruleset.]rule \
  -message message_text
```

Arguments

- rule Specify the *rule* to deselect using the rule label. Optionally specify the *language* for rules that apply to both VHDL and Verilog. Also, optionally specify the *policy* and *ruleset*.
- message Specify the message text to be used for the specified rule. Enclose messages longer than one word in double quotes.

Example

The following example sets the message for `RULE_1` to “This is a new message”. This command does not return a value when it completes successfully.

```
leda> rule_set_message -rule RULE_1 -message "This is a new message"
```

rule_set_parameter

Use the `rule_set_parameter` command to change the value node for a rule. For the prepackaged rules that come with Leda, there is a set of predefined macros that you can use to access and change the value that a rule is constraining (see [“Predefined Macros for Prepackaged Rules” on page 243](#)).

For custom rules that you write, you need to build a macro into your VerSL or VRSL source code for a rule in the form:

```
<label>_<paraName>
```

in order to be able to later modify the value of the parameter for that rule using the `rule_set_parameter` command.

When this command executes successfully, it does not return a value. You can confirm that the parameter was set as you wanted using the [propagate](#) command.

Syntax

```
rule_set_parameter -rule label -parameter (label | macro_name) \
  -value value
```

Arguments

- rule Specify the *label* for the rule whose value node you want to change.

- parameter Specify the parameter *label* or predefined *macro_name* for the parameter that you want to set. In general, you use parameter labels to modify value nodes for custom rules you created and macro names to modify value nodes for prepackaged rules. (See “[Predefined Macros for Prepackaged Rules](#)” on [page 243](#).).
- value Specify the *value* for the parameter.

Example

For example, rule B_4203 in the Leda General Coding Guidelines policy concerns module names. In fact, this rule makes sure that all module names in your design end in “_MOD”. If your design team has a different convention, you can easily change the value that the rule enforces, using the `rule_set_parameter` command, as follows:

```
leda> rule_set_parameter -rule B_2403 -parameter MODULE_NAME \
      -value "_MODULE$"
```

After you run this command, rule B_2403 enforces a new naming convention on module names in your design. Note the `MODULE_NAME` string in this example, which is a predefined macro that you can use with specific prepackaged rules where it makes sense (rules about module names).



Note

You can also use the Leda Rule Wizard in the GUI to change value nodes for rules (**Check > Configure** from main window).

Defining Multiple Values into a Parameter

Following is the syntax for defining multiple values into a parameter if the rule is a block-level rule.

Syntax

```
rule_set_parameter -rule label -parameter (label | macro_name) \
      -value {reg_exp}
```

Example

For example, rule VER_1_1_1_7 in the VER_STARC_DSG policy concerns active low signal names. This rule makes sure that all active low signals in your design end with either “_X” or “_N”. If your design team has a different convention, you can easily change the value that the rule enforces, using the `rule_set_parameter` command, as follows:

```
leda> rule_set_parameter -rule VER_1_1_1_7 -parameter SIGNAL_NAME \
      -value {_X$|_N$}
```

After you run this command, rule VER_1_1_1_7 enforces a new naming convention on signals that are active low in your design.

Following is the syntax for defining multiple values into a parameter if the rule is a netlist rule.

Syntax

```
rule_set_parameter -rule label -parameter (label | macro_name) \
    -value {value_1 value_2 value_3 ... value_n}
```

Example

For example, multiple values for TIE_OFF_CELLS parameter of rule NTL_CON17 can be specified as follows:

```
leda> rule_set_parameter -rule NTL_CON17 -parameter TIE_OFF_CELLS \
    -value {tohlslxl tohlxl tohsxl tohx1 tolsxl tolxl}
```

Predefined Macros for Prepackaged Rules

Following are predefined rule-specific, rule_set_parameter commands that you can use as a reference or cut-and-paste to your Tcl shell in the Leda environment. The values shown in these commands are the existing defaults. Change the parameter's value argument to your new setting before checking any of these rules:

```
rule_set_parameter -rule A_3C_R -parameter SYNCHRONIZER_FF_NUMBER -value {1}
rule_set_parameter -rule B_1006 -parameter MODULE_NAME -value {^TOP}
rule_set_parameter -rule B_1202 -parameter NB_MAX_CLOCKS -value {1}
rule_set_parameter -rule C_1200 -parameter NB_MAX_CLOCKS -value {1}
rule_set_parameter -rule C_1202 -parameter SYNCHRONIZER_FF_NUMBER -value {1}
rule_set_parameter -rule C_1204 -parameter UNIT_NAME -value {^CKGEN$}
rule_set_parameter -rule B_1405 -parameter NB_MAX_ASYNC_RESETS -value {1}
rule_set_parameter -rule B_1406 -parameter NB_MAX_SYNC_RESETS -value {1}
rule_set_parameter -rule B_1409 -parameter NB_MAX_ASYNC_RESETS -value {1}
rule_set_parameter -rule B_1410 -parameter NB_MAX_SYNC_RESETS -value {1}
rule_set_parameter -rule C_1400 -parameter NB_MAX_RESETS -value {1}
rule_set_parameter -rule C_1402 -parameter UNIT_NAME -value {RSTGEN}
rule_set_parameter -rule B_3606 -parameter STATE_NAME -value {_cs$}
rule_set_parameter -rule B_3608 -parameter NB_MAX_STATES -value {40}
rule_set_parameter -rule B_4200 -parameter ENTITY_NAME -value {_ENT$}
rule_set_parameter -rule B_4201 -parameter FILENAME -value {<entity>.vhd}
rule_set_parameter -rule B_4202 -parameter ARCHITECTURE_NAME -value {_ARC$}
rule_set_parameter -rule B_4203 -parameter MODULE_NAME -value {_MOD$}
rule_set_parameter -rule B_4204 -parameter FILENAME \
    -value {<architecture>.vhd}
rule_set_parameter -rule B_4205 -parameter FILENAME -value {<module>.v}
rule_set_parameter -rule B_4206 -parameter PACKAGE_NAME -value {_PACK$}
rule_set_parameter -rule B_4207 -parameter FILENAME -value {<package>.vhd}
rule_set_parameter -rule B_4208 -parameter FILENAME \
    -value {<package>_body.vhd}
rule_set_parameter -rule B_4209 -parameter CONFIGURATION_NAME -value {_CONF$}
```

```

rule_set_parameter -rule B_4210 -parameter FILENAME \
  -value {<configuration>.vhd}
rule_set_parameter -rule B_4211 -parameter SIGNAL_NAME -value {^S}
rule_set_parameter -rule B_4212 -parameter VARIABLE_NAME -value {^V}
rule_set_parameter -rule B_4213 -parameter CONSTANT_NAME -value {^C}
rule_set_parameter -rule B_4214 -parameter COMPONENT_NAME -value {^COMP}
rule_set_parameter -rule B_4215 -parameter TYPE_NAME -value {^T}
rule_set_parameter -rule B_4216 -parameter SUBTYPE_NAME -value {^ST}
rule_set_parameter -rule B_4217 -parameter FUNCTION_NAME -value {^F}
rule_set_parameter -rule B_4218 -parameter PROCEDURE_NAME -value {^P}
rule_set_parameter -rule B_4219 -parameter INSTANCE_NAME -value {^U}
rule_set_parameter -rule B_4220 -parameter BLOCK_NAME -value {_BLOCK$}
rule_set_parameter -rule B_4221 -parameter GENERATE_NAME -value {_GEN$}
rule_set_parameter -rule B_4222 -parameter ALWAYS_NAME -value {_ALW$}
rule_set_parameter -rule B_4223 -parameter PROCESS_NAME -value {_PROC$}
rule_set_parameter -rule B_4224 -parameter PRIMITIVE_NAME -value {^P}
rule_set_parameter -rule B_4225 -parameter REGISTER_NAME -value {_r$}
rule_set_parameter -rule B_4226 -parameter NET_NAME -value {^w}
rule_set_parameter -rule B_4227 -parameter INPUT_PORT_NAME -value {_in$}
rule_set_parameter -rule B_4228 -parameter OUTPUT_PORT_NAME -value {_out$}
rule_set_parameter -rule B_4229 -parameter INOUT_PORT_NAME -value {_inout$}
rule_set_parameter -rule B_4230 -parameter TASK_NAME -value {^T}
rule_set_parameter -rule B_4231 -parameter INITIAL_NAME -value {_INIT$}
rule_set_parameter -rule B_4400 -parameter LATCH_INPUT_NAME -value {_d$}
rule_set_parameter -rule B_4401 -parameter LATCH_OUTPUT_NAME -value {_q$}
rule_set_parameter -rule B_4402 -parameter FF_INPUT_NAME -value {_d$}
rule_set_parameter -rule B_4403 -parameter FF_OUTPUT_NAME -value {_r$}
rule_set_parameter -rule B_4404 -parameter CLK_NAME -value {^clk}
rule_set_parameter -rule B_4405 -parameter ASYNC_RST_NAME -value {^rst}
rule_set_parameter -rule B_4406 -parameter SYNC_RST_NAME -value {^rst}
rule_set_parameter -rule B_4407 -parameter TRISTATE_NAME -value {_z$}
rule_set_parameter -rule VER_1_1_1_1 -parameter FILENAME \
  -value {^<module>.v$}
rule_set_parameter -rule VER_1_1_1_7 -parameter SIGNAL_NAME -value {_X$|_N$}
rule_set_parameter -rule VER_1_1_1_8 -parameter INSTANCE_NAME \
  -value {^<module>$\|^<module>\[0-9_\]+}$}
rule_set_parameter -rule VER_1_1_1_9A -parameter MAX_TOP_NAME_LENGTH \
  -value {16}
rule_set_parameter -rule VER_1_1_1_9C -parameter MAX_TOP_PORT_NAME_LENGTH \
  -value {16}
rule_set_parameter -rule VER_1_1_2_1A -parameter MIN_MODULE_NAME_LENGTH \
  -value {2}
rule_set_parameter -rule VER_1_1_2_1B -parameter MAX_MODULE_NAME_LENGTH \
  -value {32}
rule_set_parameter -rule VER_1_1_2_1C -parameter MIN_INSTANCE_NAME_LENGTH \
  -value {2}
rule_set_parameter -rule VER_1_1_2_1D -parameter MAX_INSTANCE_NAME_LENGTH \
  -value {32}
rule_set_parameter -rule VER_1_1_3_3A -parameter MIN_CHARS -value {2}
rule_set_parameter -rule VER_1_1_3_3B -parameter MAX_CHARS -value {40}
rule_set_parameter -rule VER_1_1_4_1 -parameter FILENAME \
  -value {^.+h$|^.+vh$|^.+inc$|^.+ht$|^.+tsk$}
rule_set_parameter -rule VER_1_1_4_2 -parameter PARAMETER_NAME -value {^P}

```

```

rule_set_parameter -rule VER_1_1_5_1A -parameter FF_OUTPUT_NAME \
    -value {_REG$_|_reg$}
rule_set_parameter -rule VER_1_1_5_1B -parameter FF_OUTPUT_NAME \
    -value {^<clock>|<clock>$}
rule_set_parameter -rule VER_1_1_5_2A -parameter CLK_NAME -value \
    {^CLK$_|^CK$_|^CLK\[0-9_\]+$_|^CK\[0-9_\]+$_|^CLK\[a-zA-Z0-9_\]$_|^CLK\[a-zA-Z0-9_\] \
    \[a-zA-Z0-9_\]$_|^CLK\[a-zA-Z0-9_\] \[a-zA-Z0-9_\] \[a-zA-Z0-9_\]$_|^CK\[a-zA-Z0-9_\] \
    \[a-zA-Z0-9_\]$_|^CK\[a-zA-Z0-9_\] \[a-zA-Z0-9_\]$_|^CK\[a-zA-Z0-9_\] \[a-zA-Z0-9_\] \
    \[a-zA-Z0-9_\]$_}
rule_set_parameter -rule VER_1_1_5_2B -parameter RST_NAME -value \
    {^RST$_|^RST\[0-9_\]+$_|^RST\[a-zA-Z0-9_\]$_|^RST\[a-zA-Z0-9_\] \[a-zA-Z0-9_\]$_ \
    \|^RST\[a-zA-Z0-9_\] \[a-zA-Z0-9_\] \[a-zA-Z0-9_\]$_}
rule_set_parameter -rule VER_1_2_1_1A -parameter NB_MAX_CLOCKS -value {1}
rule_set_parameter -rule VER_1_3_2_1 -parameter UNIT_NAME -value {GENRST}
rule_set_parameter -rule VER_1_4_1_1 -parameter UNIT_NAME -value {GENCLK}
rule_set_parameter -rule VER_1_4_4_1 -parameter NB_MAX_CLOCKS -value {1}
rule_set_parameter -rule VER_1_5_1_1 -parameter SYNCHRONIZER_FF_NUMBER \
    -value {1}
rule_set_parameter -rule VER_1_6_4_3 -parameter NB_MAX_PORTS -value {200}
rule_set_parameter -rule VER_2_5_1_4 -parameter MAX_DRIVERS -value {5}
rule_set_parameter -rule VER_2_6_1_3 -parameter NB_MAX_OUTPUTS -value {5}
rule_set_parameter -rule VER_2_8_2_1 -parameter NB_MAX_BITS -value {16}
rule_set_parameter -rule VER_2_8_2_2 -parameter NB_MAX_CASE_ITEMS \
    -value {100}
rule_set_parameter -rule VER_2_11_1_4 -parameter NB_MAX_STATES -value {40}
rule_set_parameter -rule VER_3_2_5_2 -parameter NB_MAX_BITS -value {15}
rule_set_parameter -rule VER_3_5_3_1A -parameter HEADER_CONTENT \
    -value {FILENAME}
rule_set_parameter -rule VER_3_5_3_1B -parameter HEADER_CONTENT -value {TYPE}
rule_set_parameter -rule VER_3_5_3_1C -parameter HEADER_CONTENT \
    -value {FUNCTION}
rule_set_parameter -rule VER_3_5_3_1D -parameter HEADER_CONTENT -value {edit}
rule_set_parameter -rule VER_3_5_3_1E -parameter HEADER_CONTENT \
    -value {Author}
rule_set_parameter -rule VER_3_5_3_1F -parameter HEADER_CONTENT -value {Date}
rule_set_parameter -rule VHD_1_1_1_1 -parameter FILENAME \
    -value {^<entity>.vhd$}
rule_set_parameter -rule VHD_1_1_1_7 -parameter SIGNAL_NAME -value {_X$_|_N$_}
rule_set_parameter -rule VHD_1_1_1_8 -parameter INSTANCE_NAME -value \
    {^U<entity>$|^U<entity>$|^U<entity>_\[0-9\]$_|^U<entity>_\[0-9\]$_|^U<entit
y>_\[0-9\] \[0-9\]$_|^U<entity>_\[0-9\] \[0-9\]$_|^U<entity>_\[0-9\] \[0-9\] \
    \[0-9\]$_|^U<entity>_\[0-9\] \[0-9\] \[0-9\] \[0-9\]$_}
rule_set_parameter -rule VHD_1_1_1_9A -parameter MAX_ENTITY_NAME_LENGTH \
    -value {16}
rule_set_parameter -rule VHD_1_1_1_9C -parameter MAX_PORT_NAME_LENGTH \
    -value {16}
rule_set_parameter -rule VHD_1_1_2_1A -parameter MIN_ENTITY_NAME_LENGTH \
    -value {2}
rule_set_parameter -rule VHD_1_1_2_1B -parameter MAX_ENTITY_NAME_LENGTH \
    -value {32}
rule_set_parameter -rule VHD_1_1_2_1C -parameter MIN_INSTANCE_NAME_LENGTH \
    -value {2}

```

```

rule_set_parameter -rule VHD_1_1_2_1D -parameter MAX_INSTANCE_NAME_LENGTH \
-value {32}
rule_set_parameter -rule VHD_1_1_3_3A -parameter MIN_CHARS -value {2}
rule_set_parameter -rule VHD_1_1_3_3B -parameter MAX_CHARS -value {40}
rule_set_parameter -rule VHD_1_1_4_1 -parameter FILENAME -value {_pac.vhd$}
rule_set_parameter -rule VHD_1_1_4_2B -parameter CONSTANT_NAME \
-value {^C_\|^P_}
rule_set_parameter -rule VHD_1_1_5_1A -parameter FF_OUTPUT_NAME \
-value {_REG$_\|_reg$}
rule_set_parameter -rule VHD_1_1_5_1B -parameter FF_OUTPUT_NAME \
-value {<clock>}
rule_set_parameter -rule VHD_1_1_5_2A -parameter CLK_NAME -value \
^CLK$_\|^CK$_\|^CLK\[0-9_\]+$_\|^CK\[0-9_\]+$_\|^CLK\[A-Z0-9_\]$_\|^CLK\[A-Z0-9_\]\[
A-Z0-9_\]$_\|^CLK\[A-Z0-9_\]\[A-Z0-9_\]\[A-Z0-9_\]$_\|^CK\[A-Z0-9_\]$_\|^CK\[A-Z0-
9_\]\[A-Z0-9_\]$_\|^CK\[A-Z0-9_\]\[A-Z0-9_\]\[A-Z0-9_\]$_{
rule_set_parameter -rule VHD_1_1_5_2B -parameter RST_NAME -value \
{^RST$_\|^RST\[0-9_\]+$_\|^RST\[a-zA-Z0-9_\]$_\|^RST\[a-zA-Z0-9_\]\[a-zA-Z0-9_\]$_\|^RST\[a-zA-Z0-9_\]\[a-zA-Z0-9_\]\[a-zA-Z0-9_\]$_}
rule_set_parameter -rule VHD_1_1_6_1 -parameter ARCHITECTURE_NAME \
-value {^RTL$_\|^BEH$_\|^TB$_\|^SIM$}
rule_set_parameter -rule VHD_1_2_1_1A -parameter NB_MAX_CLOCKS -value {1}
rule_set_parameter -rule VHD_1_3_2_1 -parameter UNIT_NAME -value {GENRST}
rule_set_parameter -rule VHD_1_4_1_1 -parameter UNIT_NAME -value {GENCLK}
rule_set_parameter -rule VHD_1_4_4_1 -parameter NB_MAX_CLOCKS -value {1}
rule_set_parameter -rule VHD_1_5_1_1 -parameter SYNCHRONIZER_FF_NUMBER \
-value {1}
rule_set_parameter -rule VHD_1_6_4_3 -parameter NB_MAX_PORTS -value {200}
rule_set_parameter -rule VHD_2_1_5_1 -parameter NB_MAX_LEVELS -value {5}
rule_set_parameter -rule VHD_2_3_3_1 -parameter NB_MAX_CLOCK_EDGE -value {1}
rule_set_parameter -rule VHD_2_5_1_4 -parameter MAX_DRIVERS -value {5}
rule_set_parameter -rule VHD_2_6_1_3 -parameter NB_MAX_OUTPUTS -value {5}
rule_set_parameter -rule VHD_2_6_1_4 -parameter NB_MAX_LINES -value {200}
rule_set_parameter -rule VHD_2_7_3_1 -parameter NB_MAX_LEVELS -value {5}
rule_set_parameter -rule VHD_2_8_2_2 -parameter NB_MAX_CASE_ITEMS \
-value {100}
rule_set_parameter -rule VHD_2_11_1_4 -parameter NB_MAX_STATES -value {40}
rule_set_parameter -rule VHD_3_1_4_5 -parameter LINE_LENGTH -value {110}
rule_set_parameter -rule VHD_3_5_3_1A -parameter HEADER_CONTENT \
-value {FILENAME}
rule_set_parameter -rule VHD_3_5_3_1B -parameter HEADER_CONTENT -value {TYPE}
rule_set_parameter -rule VHD_3_5_3_1C -parameter HEADER_CONTENT \
-value {FUNCTION}
rule_set_parameter -rule VHD_3_5_3_1D -parameter HEADER_CONTENT -value {edit}
rule_set_parameter -rule VHD_3_5_3_1E -parameter HEADER_CONTENT \
-value {Author}
rule_set_parameter -rule VHD_3_5_3_1F -parameter HEADER_CONTENT -value {Date}

```

rule_set_severity

Use the `rule_set_severity` command to change the severity level for a rule. This command returns a 1 (true) when it completes successfully. Otherwise, you get an error message.

Syntax

```
rule_set_severity -rule [language.][policy.][ruleset.]rule \  
-severity level
```

Arguments

- | | |
|-----------|--|
| -rule | Specify the <i>rule</i> name using the <i>language</i> , <i>policy</i> , <i>ruleset</i> , and rule label, or just specify the <i>rule</i> label. |
| -severity | Set the new severity <i>level</i> to be flagged by the tool when the specified rule is violated. Legal values include note, warning, error, and fatal. |

Example

The following example sets the severity level for rule C_1005 to NOTE. The return value of 1 at the end of the transcript means that the operation completed successfully.

```
leda> rule_set_severity -rule VERILOG.LEDA.DESIGN_STRUCTURE.C_1005 \  
-severity NOTE  
1
```

set_clock_gating_cell

Use the `set_clock_gating_cell` command to set the specified cells as clock gating cells.

Syntax

```
set_clock_gating_cells {list of cell names}
```

Arguments

- | | |
|------------|--|
| cell names | Specify the Verilog module/primitive names or VHDL entity names as <i>cell names</i> . |
|------------|--|

For more information, see the [Leda Power Rules Guide](#).

set_enable_pin

Use the `set_enable_pin` command to specify the enable pin name for a specific cell.

Syntax

```
set_enable_pin cell_name enable_pin_name
```

Arguments

cell names Specify the Verilog module/primitive names or VHDL entity names as *cell names*.

Example

This command will set EN as the enable pin for the cell IC12V.

```
leda> set_enable_pin IC12V EN
```

For more information, see the [Leda Power Rules Guide](#).

set_level_shifter

Use the `set_level_shifter` command to set the specified cells as level shifters.

Syntax

```
set_level_shifter {list of cell name}
```

Arguments

cell names Specify the Verilog module/primitive names or VHDL entity names as *cell names*.

For more information, see the [Leda Power Rules Guide](#).

set_operating_conditions

Use the `set_operating_conditions` command to set the operating conditions.

Syntax

```
set_operating_conditions [-library library] [-object_list object_list] \
  [-max max_condition] [-min min_condition] \
  [-max_library max_library] [-min_library min_library] \
  [-max_phys <max_operating_condition_name>] \
  [-min_phys <min_operating_condition_name>] \
  [-object_list object_list] [-power_domains power_domain_list] \
  [condition]
```

Arguments

-library Specify the library to search.

-max	Specify the maximum operating condition name.
-min	Specify the maximum operating condition name.
-max_library	Specify the library containing maximum operating conditions.
-min_library	Specify the library containing minimum operating conditions.
-max_phys	Specify the name of the maximum phys tech operating conditions.
-min_library	Specify the name of the minimum phys tech operating conditions.
-object_list	Specify the port/cell objects.
-power_domains	Specify the power domain objects.
condition	Specify the single operating condition name.

set_pin_voltage

Use the `set_pin_voltage` command to specify the voltage of a particular pin of a cell.

Syntax

```
set_pin_voltage cell_name pin_name voltage_float_value
```

Arguments

cell names Specify the Verilog module/primitive names or VHDL entity names as *cell names*.

Example

This command will set pin A of cell LS12V to 1.2 volts.

```
leda> set_pin_voltage LS12V A 1.2
```

For more information, see the [Leda Power Rules Guide](#).

set_power_pin

Use the `set_power_pin` command to specify the power pins of the given cell.

Syntax

```
set_power_pin [-power cell_name pin_name] [-gnd cell_name pin_name]
```

set_power_domain

Use the `set_power_domain` command to set the power domain with given instances as power regions.

Syntax

```
set_power_domain -name <name> [-always_on] <instance_list>
```

Arguments

`-name` Specify the domain name.

`-instance_list` Specify the instance list.

set_power_domain_ctrl

Use the `set_power_domain_ctrl` command to associate control signals(s) with each power domain.

Syntax

```
set_power_domain_ctrl [-name domain_name] [-signals signal /
  {signal_list} value | {value_list}
```

Arguments

`signal_list` Specifies the list of hierarchical names of ports/signals.

`value_list` Specifies the list of values for the signals at which the corresponding power domain is switched on.

Example

This command will switch on power domain POW1 when TOP.CTRL1 is equal to 1 and TOP>CTRL2 is equal to 0.

```
leda> set_power_domain_ctrl -name POW1 -signals {TOP.CTRL1 TOP.CTRL2}
      10
```

For more information, see the [Leda Power Rules Guide](#).

set_power_off_value

Some low-power methodologies impose the output of isolation cells to be set to a specific fixed value (0 or 1) when the corresponding power domain is switched off. Use the `set_power_off_value` command to specify this fixed value for any isolation cell instance.

Syntax

```
set_power_off_value    boolean_value    {isolation_cell_instance_list}
```

Example

```
leda> set_power_off_value 0 TOP.A.ISCEL3
```

For more information, see the [Leda Power Rules Guide](#).

set_power_switch

Use the `set_power_switch` command to specify the cell list as power switch cells.

Syntax

```
set_power_switch cell_name_list
```

Arguments

cell_name_list Specifies the cell list.

sizeof_collection

Use the `sizeof_collection` command to get the number of objects in a collection.

Syntax

```
sizeof_collection collection1
```

Arguments

collection1 Specifies the collection for which to get the number of objects. If an empty collection (empty string) is used for the `collection1` argument, the command returns 0.

sort_collection

Use the `sort_collection` command to sort a collection based on one or more attributes, resulting in a new, sorted collection. The sort is ascending by default.

Syntax

```
sort_collection [-descending] collection1 criteria
```

Arguments

-descending Indicates that the collection is to be sorted in reverse order. By default, the sort proceeds in ascending order.

collection1 Specifies the collection to be sorted.

`criteria` Specifies a list of one or more application or user-defined attributes to use as sort keys.

Project Tcl Command Reference

Following is command reference information for the built-in Tcl commands that you can use to manage Leda project files. To see the help for all `project_*` commands implemented in Leda, use the `help -v` switch from the Tcl prompt in the Tcl console at the bottom of the GUI or in the Tcl shell when you are not running the GUI:

```
leda> help -v project_*
```

`project_add_library`

Use the `project_add_library` command to add a logical or physical library to the current project. This command does not return a value when it completes successfully.

Syntax

```
project_add_library [-read] -library_s library_name
```

Arguments

- | | |
|-------------------------|--|
| <code>-read</code> | Get available files from this library. |
| <code>-library_s</code> | Specify the full path to the logical or physical <i>library_name</i> . |

Example

The following example adds the `lib1.vhd` library to the current project.

```
leda> project_add_library -library /u/me/work/lib1.vhd
```

project_build

Use the `project_build` command to read and compile the current project files.

Syntax

```
project_build
```

Arguments

None.

Example

The following example builds the current project. The returned value of 0 at the bottom of the transcript indicates a successful build.

```
leda> project_build
Building project...
Reading file /d/techpub1/docmaster/leda/leda403_software/test/mixed/src/
reg.v
Reading file /d/techpub1/docmaster/leda/leda403_software/test/mixed/src/
shift_reg.v
Reading file /d/techpub1/docmaster/leda/leda403_software/test/mixed/src/
stage2.v
Reading file /d/techpub1/docmaster/leda/leda403_software/test/mixed/src/
topunit.v
Compiling Module Declaration simple_reg
Compiling Module Declaration shift_reg
Compiling Module Declaration stage2
Compiling Module Declaration top
Compiling Entity Declaration MISC_LOGIC
Compiling Architecture DATAFLOW of MISC_LOGIC
Compiling Entity Declaration STAGE1
Compiling Architecture RTL of STAGE1
Building project done
0
```

project_delete

Use the delete command to delete a project from disk (by default, the current project).

Syntax

```
project_delete [-project_name project_name]
```

Arguments

-project_name If you don't specify a *project_name*, this command deletes the current project by default.

Example

The following example deletes a project that is not the current project.

```
leda> project_delete -project_name example.pro
```

project_get_all_files

Use the project_get_all_files command to get a list of all files in the specified library.

Syntax

```
project_get_all_files -work library_name -format language
```

Arguments

-work Specify the *library_name*.
-format Set the *language*. Legal values include verilog and vhd1.

Returned Values

Returns a list of library file names for the specified library name and language.

Example

The following example returns a list of Verilog files from the LEDA_WORK library for the Demo project that comes with the tool.

```
leda> project_get_all_files -work LEDA_WORK -format verilog  
/d/techpub1/docmaster/leda/leda403_software/test/mixed/src/reg.v  
/d/techpub1/docmaster/leda/leda403_software/test/mixed/src/shift_reg.v  
/d/techpub1/docmaster/leda/leda403_software/test/mixed/src/stage2.v  
/d/techpub1/docmaster/leda/leda403_software/test/mixed/src/topunit.v
```

project_get_file_attributes

Use the `project_get_file_attributes` command to get file attributes for a given language, working library, file format, directory, file extension, or any combination of these.

Syntax

```
project_get_file_attributes [-format language] [-work library_name] \  
[-file] [-directory] [-directory_file][-file_extension]
```

Arguments

-format	Specify the <i>language</i> . Legal values include verilog and vhdl.
-file	Returns a list of available files for the specified language.
-work	Specify a working <i>library_name</i> .
-directory	Returns a list of available directories that match the language specified with -format.
-directory_file	Returns a list of available files in the specified directory that match the language specified with -format.
-file_extension	Specify the file extension to look for (for example, v for Verilog and vhd or vhdl for VHDL).

Returned Values

Returns a list of files for the specified language, library, directory, files, or file extension.

Example

The following example returns a list of source files from the LEDA_WORK library for the Demo project that comes with the tool. Because this is a mixed-language project, you see file extensions for both VHDL and Verilog in the command results.

```
leda> project_get_file_attributes -work LEDA_WORK -file_extension  
.vhd .vhdl .v .ve .inc
```


project_get_library_attribute

Use the `project_get_library_attribute` command to get information about working libraries and resource libraries for your project.

Syntax

```
project_get_library_attribute [-format language] attribute_name
```

Arguments

<code>-format</code>	Set the <i>language</i> . Legal values include <code>verilog</code> and <code>vhdl</code> . Default is <code>all</code> .
<i>attribute_name</i>	Specify the <i>attribute_name</i> of interest. Legal values include <code>mapping</code> , <code>top_level_library</code> , <code>resource</code> , <code>library_directory</code> , <code>library_file</code> , <code>library_extension</code> , <code>checklib</code> , and <code>append</code> .

Returned Values

Returns information for the specified *language* and *attribute_name*.

Example

The following example returns the name of the top-level library in the Demo project that ships with the tool:

```
leda> project_get_library_attribute top_level_library
LEDA_WORK
```

project_get_option_attribute

Use the `project_get_option_attribute` command to get information about optional analyzer settings that may be in effect for the current project.

Syntax

```
project_get_option_attribute [-format language] -name attribute_name
```

Arguments

- | | |
|---------|--|
| -format | Set the <i>language</i> . Legal values include <code>verilog</code> and <code>vhdl</code> . Default is <code>all</code> . |
| -name | Specify the <i>attribute_name</i> of interest. Legal values include <code>no_semantic_exception</code> , <code>nocase</code> , <code>translate_directive</code> , <code>severity</code> , <code>macro</code> , <code>include</code> , and <code>version</code> . |

Example

The following example returns a value of “warning” to let you know that messages from the analyzer of this severity and above will be printed.:

```
leda> project_get_option_attribute -format verilog -name severity
warning
```

project_get_ports

Use the `project_get_ports` command to get a list all inout/out ports for a given unit.

Syntax

```
project_get_ports -work library_name -top unit_name
```

Arguments

- | | |
|-------|---|
| -work | Specify the <i>library_name</i> . |
| -top | Specify the top-level <i>unit_name</i> in the design hierarchy. |

Example

The following example gets a list of all inout/out ports for the `LEDA_WORK` library in the Demo project that comes with the tool:

```
leda> project_get_ports -work LEDA_WORK -top top
clk din enb rst
```

project_get_top_units

Use the `project_get_top_units` command to get all potential top units from a given library.

Syntax

```
project_get_top_units -work library_name
```

Arguments

`-work` Specify the working *library_name*.

Example

The following example returns a list of potential top units from the LEDA_WORK working library:

```
leda> project_get_top_units -work LEDA_WORK
top DATAFLOW/MISC_LOGIC MISC_LOGIC RTL/STAGE1 STAGE1 shift_reg
simple_reg stage2
```

project_get_unit_kinds_from_library

Use the `project_get_unit_kinds_from_library` command to get a list of all unit kinds from a given working library.

Syntax

```
project_get_unit_kinds_from_library working_library
```

Arguments

`library` Specify the *working_library* name.

Example

The following example returns a list of all unit kinds from the LEDA_WORK library in the Demo project.

```
leda> project_get_unit_kinds_from_library LEDA_WORK
MODULE ENTITY
```

project_get_units_from_file

Use the `project_get_units_from_file` command to get a list of all units in a given file.

Syntax

```
project_get_units_from_file file_name [-format language]
```

Arguments

<i>file_name</i>	Specify the <i>file_name</i> .
<code>-format</code>	Set the <i>language</i> . Legal values include verilog and vhdl. Default is all.

Returned Values

Returns a list of all units from the specified file.

Example

The following example returns a list of Verilog units instantiated in the `top.v` file.

```
leda> project_get_units_from_file top.v -format verilog
```

project_get_units_from_library

Use the `project_get_units_from_library` command to get a list of unit names, unit kinds, compilation dates, obsolete flag statuses, file names, and number of lines for a given library.

Syntax

```
project_get_units_from_library library_name
```

Arguments

library_name Specify the *library_name*.

Returned Values

Returns a information about the specified library.

Example

The following example returns a list of unit names, kinds, compilation dates, obsolete flag statuses, file names, and number of lines for the LEDA_WORK library.

```
leda> project_get_units_from_library LEDA_WORK
{MISC_LOGIC;ENTITY;1084834151;;/d/techpub1/docmaster/leda/
leda403_software/test/mixed/src/misc_logic.vhd;1}
{STAGE1;ENTITY;1084834151;;/d/techpub1/docmaster/leda/leda403_software/
test/mixed/src/stage1.vhd;1} {DATAFLOW/
MISC_LOGIC;ARCHITECTURE;1084834151;;/d/techpub1/docmaster/leda/
leda403_software/test/mixed/src/misc_logic.vhd;13} {RTL/
STAGE1;ARCHITECTURE;1084834151;;/d/techpub1/docmaster/leda/
leda403_software/test/mixed/src/stage1.vhd;15}
{simple_reg;MODULE;1084834149;;/d/techpub1/docmaster/leda/
leda403_software/test/mixed/src/reg.v;1} {shift_reg;MODULE;1084834149;;/
d/techpub1/docmaster/leda/leda403_software/test/mixed/src/
shift_reg.v;13} {stage2;MODULE;1084834149;;/d/techpub1/docmaster/leda/
leda403_software/test/mixed/src/stage2.v;36} {top;MODULE;1084834149;;/d/
techpub1/docmaster/leda/leda403_software/test/mixed/src/topunit.v;53}
```

project_get_working_libraries

Use the `project_get_working_libraries` command to get the working libraries for the current project.

Syntax

```
project_get_working_libraries
```

Arguments

None.

Returned Values

Returns a list of working libraries for the current project.

Example

The following example returns the working libraries for the Demo project after the user added one:

```
leda> project_get_working_libraries
LEDA_WORK EIGHTADD
```

project_new

Use the `project_new` command to create an empty project and set it as the default for the current session.

Syntax

```
project_new [-quiet] [project_name]
```

Arguments

<code>-quiet</code>	Don't display info messages.
<i>project_name</i>	Specify the <i>project_name</i> for the new project. If you don't specify a <i>project_name</i> , Leda creates a project called <code>leda.pro</code> by default.

Example

The following example creates a new Leda project called DAVEY. Note that all new projects you create in Leda are given a `.pro` extension by convention.

```
leda> project_new DAVEY
New project DAVEY.pro done
/d/techpub1/docmaster/leda/leda403_software/DAVEY.pro
```

project_open

Use the `project_open` command to open a project.

Syntax

```
project_open [-quiet] [-create project_name] [-project project_name]
```

Arguments

<code>-quiet</code>	Don't display info messages.
<code>-create</code>	Specify the <i>project_name</i> if it doesn't already exist. Note that all new projects you create in Leda are given a <code>.pro</code> extension by convention.
<code>-project</code>	Specify the <i>project_name</i> for an existing project. If you don't specify a <i>project_name</i> , Leda opens the Demo project that comes with the tool by default (<code>demo.pro</code>).

Example

The following example creates and opens a new project named DAVEY.

```
leda> project_open -create DAVEY
New project DAVEY.pro done
DAVEY
```

project_quit

Use the `project_quit` command to exit a project.

Syntax

```
project_quit
```

Arguments

None.

Example

The following example exits the current project. The returned value of 0 indicates that you quit the project successfully.

```
leda> project_quit
0
```

project_read

Use the `project_read` command to display the contents of a project.

Syntax

```
project_read [project_name]
```

Arguments

project_name Specify the *project_name*. If you don't specify a *project_name*, Leda reads a project called `leda.pro` by default.

Example

The following example displays information about the `demo.pro` project that comes with the tool:

```
leda> project_read demo.pro
project_specify_files { {$LEDA_PATH/test/mixed/src} }
checker_set_design_constraints -nowarning -clockdump -top top
checker_set_design_constraints -top { LEDA_WORK top } -nowarning
-clockdump
```

project_record_cmd

Use the `project_record_cmd` command to save a sequence of Tcl project commands for a Leda project.

Syntax

```
project_record_cmd
```

Arguments

None.

Example

The following example records a sequence of Tcl commands and saves them into a Tcl project file.

```
leda> project_new test.pro
leda> proc test {} {project_record_cmd}
leda> test
leda> project_specify_files $::env(PROTON_ROOT)/test/mixed/src/reg.v
leda> project_build
leda> project_save
leda> project_quit
```


project_remove_file

Use the `project_remove_file` command to remove a file from the current project, and optionally from the disk too.

Syntax

```
project_remove_file file_name -work library_name \  
[-format language] [-delete]
```

Arguments

<i>file_name</i>	Specify the <i>file_name</i> to remove.
-work	Specify the working <i>library_name</i> for the file.
-format	Set the <i>language</i> . Legal values include verilog and vhdl. Default is all.
-delete	Also deletes the file from the disk.

Example

The following example removes the `reg.v` file from the `MY_LIB` working library. This command does not return a value when it completes successfully.

```
leda> project_remove_file -file reg.v -work MY_LIB
```

project_remove_library

Use the `project_remove_library` command to remove a library from the current project, and optionally from the disk too.

Syntax

```
project_remove_library [-delete] library_name
```

Arguments

-delete	Also deletes the library from the disk.
<i>library_name</i>	Specify the <i>library_name</i> to remove.

Example

The following example removes the `MY_LIB` library from the current project. This command does not return a value when it completes successfully.

```
leda> project_remove_library MY_LIB
```

project_save

Use the `project_save` command to save a project.

Syntax

```
project_save
```

Arguments

None.

Example

The following example saves the current project. The return value of 0 indicates that the project was saved successfully.

```
leda> project_save
0
```

project_specify_files

Use the `project_specify_files` command to specify files or directories containing HDL design files that you want to add to your project.

Syntax

```
project_specify_files directory_or_file [-file_extension {extensions}] \
  [-work library_name][-format language]
```

Arguments

<i>directory_or_file</i>	Specify a <i>directory_or_file</i> name that you want to add to your project.
-file_extension	If you specified a directory for the <i>directory_or_file</i> argument, use this option to specify the file <i>extensions</i> that you want to include (e.g., v, vhd, etc.).
-work	Specify the working <i>library_name</i> .
-format	Set the <i>language</i> . Legal values include verilog and vhdl. Default is all.

Example

The following example adds the `reg.v` file the `MY_LIB` working library.

```
leda> project_specify_files $LEDA_PATH/test/mixed/src/reg.v -work MY_LIB
Reading file reg.v (no check)
```

project_specify_libraries

Use the `project_specify_libraries` command to specify working libraries and resource libraries that you want to add to your project.

Syntax

```
project_specify_libraries [-format language]\
  [-mapping {logical_name physical_name}] \
  [-top_level_library {logical_name | physical_name}] \
  [-resource {logical_name physical_name}] [-append]\
  [-library_directory {library_directory}] \
  [-library_file {library_file}] \
  [-library_extension {library_extension}] \
  [-checklib [{library_dir | library_file}] \
  [-format language]
```

Arguments

- | | |
|--------------------|--|
| -format | Set the <i>language</i> . Legal values include vhdl and verilog. Default is all. |
| -mapping | Specify the mapping between the <i>logical</i> and <i>physical</i> names of the library. |
| -top_level_library | Specify the <i>logical</i> or <i>physical</i> name for the top-level library in the design. |
| -resource | For VHDL only, specify the name of a resource library. You need to specify a logical name for the library. The resource library you specify with this option replaces the existing resource library list. If you want to add a resource library to the existing list, use the <code>-append</code> switch too.
Note: the <i>physical_name</i> must be specified as full path. |
| -append | For VHDL only. Use with the <code>-resource</code> option to add the resource library name to the existing list of resource libraries. |
| -library_directory | For Verilog only, specify the name of the <i>library_directory</i> . |
| -library_file | For Verilog only, specify the name of a <i>library_file</i> . |
| -library_extension | For Verilog only, specify the extension for files in the library. |
| -checklib | For Verilog only, specify library directories or files to be checked. |

Example

The following example maps the `my_lib1` logical library to the `golden.v` physical library and identifies the top-level-library as `top_lib.v`.

```
leda> project_specify_libraries -format verilog \  
      -mapping {my_lib1 golden.v} \  
      -top_level_library {top_lib.v} \  
      -library_directory {/u/me/my_lib1} \  
      -library_extension {.v}
```

project_specify_name

Use the `project_specify_name` command to set a name for a Leda project.

Syntax

```
project_specify_name project_name
```

Arguments

project_name Use the *project_name* argument to set the name for the project.

Example

The following example sets up a name of `proj1` for the current project:

```
leda> project_specify_name proj1  
/d/techpub1/docmaster/leda/leda403_software/proj1.pro
```

project_specify_options

Use the `project_specify_options` command to specify the analyzer options that you want Leda to use.

Syntax

```
project_specify_options -format language [-no_semantic_exception] \
  [-nocase] [-translate_directive] [-severity level] \
  [-macro {macro [=val]}] [-include {directory_name}] \
  [-version version]
```

Arguments

- format Set the *language*. Legal values include vhdl and verilog. Default is all.
- no_semantic_exception Use this switch to disable semantic exceptions in the analyzer.
- nocase Use this switch to disable case sensitivity.
- translate_directive Use this switch to make the analyzer honor Synopsys `translate_off` and `translate_on` pragmas in your HDL source files.
- severity Specify the severity *level* for which you want to see messages from the analyzer. You get the selected severity level and above. Legal values include note, warning, error, and fatal.
- macro Verilog only. Specify the value (*val*) for a *macro* previously defined in your custom or prepackaged rules.
- include Set the *directory_name* to search for include files.
- version Set the *version* for the analyzer. Legal values include 95, 01 (Verilog 2001), and 03 (SystemVerilog) for Verilog, and 87 and 93 for VHDL.

Example

The following example specifies that the Verilog analyzer use the Verilog 95 standard for analysis. This command does not return a value when it completes successfully.

```
leda> project_specify_options -format verilog -version 95
```

project_update

Use the `project_update` command to update the current project if you made changes to any of the HDL files that you are testing with Leda.

Syntax

```
project_update [-force]
```

Arguments

-force Use this switch to force a rebuild of the entire design, not just the parts that changed.

Example

The following example updates the current project. The return value of 0 indicates that the project updated successfully.

```
leda> project_update  
0
```

Checker Tcl Command Reference

Following is command reference information for the built-in Tcl commands that you can use to run the Leda Checker. These commands allow you to develop and store sequences of actions for unattended runs with the tool, such as regression tests. To see the help for all checker_* commands implemented in Leda, use the help -v switch from the Tcl prompt in the Tcl console at the bottom of the GUI or in the Tcl shell when you are not running the GUI:

```
leda> help -v check_*
```

You can store your Leda Tcl commands in a .tcl file, and source that file from the Leda Tcl shell to implement your stored procedures (see [“Sourcing a Tcl Script in Leda” on page 188](#)).

check

Use the check command to run the tool on the current project (HDL source files) using the specified run options. You must have the appropriate rules selected for checking in your configuration (block, chip, netlist) in order to get any results. For example, to run the netlist checker you must have some netlist checker rules selected. If you execute the check command without specifying any options, Leda runs the Checker using the last values for any options already set in that session.

Syntax

```
check [-format language] [-work library_name] [-block] [-chip] \
  [-netlist] [-sdc] [-nocheck] [-propagate] [-config file_name] \
  [-policy policy_name] [-ruleset ruleset_name] [-rule rule_name] \
  [-nowarning] [-top {[library_name] top_unit_name}] \
  [-noelaboration] [-case_analysis_file file_name] [-nohierdump] \
  [-clockdump] [-test_clk_rising clock_name] \
  [-test_clk_falling clock_name] [-test_async reset_name] \
  [-test_async_inverted reset_name] [-max_violations number] \
  [-nohierdump limit] [rule_name] \
  [+max_case+<val>] \
  [+max_casexz+<val>]
```

Arguments

- format Selects the *language* for rule checking. Valid values include verilog and vhdl. If you don't specify either option, Leda checks selected rules for both Verilog and VHDL by default.
- work Specify a working *library_name*.

- `-block` All checks are enabled by default starting with version 4.1. Use the `-block` switch if you want to run only block-level checks. These are unit-wide checks (for example, language-based checks).
- `-chip` All checks are enabled by default starting with version 4.1. Use the `-chip` switch if you want to run only chip-level checks. These are unit-wide checks that use hardware inference at the RTL level (for example, all flip-flops in the design must be active on the rising edge).
- `-netlist` All checks are enabled by default starting with version 4.1. Use the `-netlist` switch if you want to run only netlist checks. These checks run on the gate-level design netlist.
- `-sdc` All checks are enabled by default starting with version 4.1. Use the `-sdc` switch if you want to run only SDC checks. For more information, see [“Using the SDC Checker” on page 133](#).
- `-nocheck` Elaborates the design but doesn't run any rule checks.
- `-propagate` Enables constant propagation. See the `-case_analysis_file` option for this command.
- `-config` Specify the full path to the configuration file that you want to use for this run with the tool.
- `-policy` Specify a *policy_name* that you want to check. If you use this option, Leda ignores rule selections specified in your current configuration.
- `-ruleset` Specify a *ruleset_name* that you want to check. If you use this option, Leda ignores rule selections specified in your current configuration.
- `-rule` Specify a *rule_name* for a rule that you want to check. If you use this option, Leda ignores rule selections specified in your current configuration.
- `rule_name` Same as `-rule` (above).
- `-nowarning` Only relevant for chip-level checks. Use the `-nowarning` switch to suppresses warning messages generated during compilation and elaboration.
- `-top` Use the `-top` option to specify the *top_unit_name* or *library_name* for your design hierarchy. This is required in order to check for chip-level rule violations.

- noelaboration** Do not force re-elaboration if it is not necessary.
- case_analysis_file** Use the `-case_analysis_file` option to point Leda to an ASCII text file containing `set_case_analysis` commands that specify constant values for primary inputs or internal signals used in constant propagation. For more information, see [“Propagating Constants” on page 96](#).
- nohierdump** Use the `-nohierdump` switch to turn off generation of the hierarchy browsing database. Optionally, also specify a *limit* for the hierarchy dump instead of a full disable. This speeds up tool performance, but disables the hierarchy browser in the GUI after a Checker run.
- clockdump** Use the `-clockdump` switch to enable use of the Clock and Reset Tree browsers when you load a log file into the Error Viewer after checking your design. Note that this switch can slow Leda’s performance when checking large netlists. See [“Using the Clock and Reset Tree Browsers” on page 126](#).
- test_clk_rising** Use the `-test_clk_rising` option to specify test clock signal *clock_name* and specify that the first edge in this clock’s cycle is the rising edge. With RTLDR©, this corresponds to the following command:
- ```
create_test_clk CLK -w{N1 N1+N2}
```
- In Leda’s DFT checks, no delays are taken into account. Leda always assumes that the test clock period is 100 ns and the strobe point occurs at 95 ns (default RTLDR© values). Leda also assumes that all test clock events occur before this strobe point.
- Note:** The `-test_clk_rising` option expects the *CLK* argument to be a primary I/O name, not a hierarchical name including the top module name. Internal signals are not allowed.
- test\_clk\_falling** Use the `-test_clk_falling` option to specify test clock signal *clock\_name* and specify that the first edge in this clock’s cycle is the falling edge. With RTLDR©, this corresponds to the following command:

```
create_test_clk CLK -w{N1 N1-N2}
```

In Leda's DFT checks, no delays are taken into account. Leda always assumes that the test clock period is 100 ns and the strobe point occurs at 95 ns (default RTLDR© values). Leda also assumes that all test clock events occur before this strobe point.

**Note:** The `-test_clk_falling` option expects the *CLK* argument to be a primary I/O name, not a hierarchical name including the top module name. Internal signals are not allowed.

- `-test_async` Use the `-test_asynch` option to specify the test reset signal *reset\_name* and indicate that it is active on "1" and has a hold value of "0" during the scan shift phase. With RTLDR©, this corresponds to the following command:
- ```
set_signal_type test_asynch RST
```
- `-test_async_inverted` Use the `-test_asynch_inverted` option to specify the test reset signal *reset_name* and specify that it is active on "0" and has a hold value of "1" during the scan shift phase. With RTLDR©, this corresponds to the following command:
- ```
set_signal_type test_asynch_inverted RST
```
- `-max_violations` Use the `-maxviolations` option to set the maximum number of violations per selected rule that Leda flags. The default is 100.
- `+max_case+<val>` Use the `+max_case` option to specify the maximum width of a case expression in a case statement. The default value is 8.
- `+max_casexz+<val>` Use the `+max_casexz` option to specify the maximum width of a case expression in a casex/casexz statement. The default value is 8.

## Example

The following example runs all chip-level and block-level rules selected in the current configuration:

```
leda> check top top
Executing elaboration of top unit top ...
Elaboration of top unit top completed.
Dumping the design hierarchy...
Design hierarchy dump completed.
Executing chip-level checks on design top ...
Chip-level checks on design top completed.
Executing block-level checks on library LEDA_WORK ...
Executing block-level checks on unit LEDA_WORK.simple_reg ...
Executing block-level checks on unit LEDA_WORK.shift_reg ...
Executing block-level checks on unit LEDA_WORK.stage2 ...
Executing block-level checks on unit LEDA_WORK.top ...
Executing block-level checks on library LEDA_WORK ...
Executing block-level checks on unit LEDA_WORK.MISC_LOGIC ...
Executing block-level checks on unit LEDA_WORK.STAGE1 ...
Executing block-level checks on unit LEDA_WORK.DATAFLOW/MISC_LOGIC ...
Executing block-level checks on unit LEDA_WORK.RTL/STAGE1 ...
```

## checker\_get\_design\_constraints

Use the `checker_get_design_constraints` command to see information that the Checker is using for your runs with the tool. These options control the behavior of the tool and can affect runtime performance and the features that are enabled. You must specify an attribute name for this command to work.

### Syntax

```
checker_get_design_constraints attribute_name
```

### Arguments

*attribute\_name* Specify an *attribute\_name*. Legal values for *attribute\_name* include `blast`, `nowarning`, `top`, `noelaboration`, `constraint_file`, `nohierdump`, `maxhierdump`, `forcehierdump`, `test_clk_rising`, `test_clk_falling`, `test_async`, `test_async_inverted`, `max_violations`, and `nomaxviolations`. For information on what these attributes control during a Checker run, see [propagate](#).

### Example

The following example returns the value set for `max_violations`, which sets the maximum number of errors that Leda flags for any one rule selected for checking. The current setting for this design is 100, which is the default:

```
leda> checker_get_design_constraints max_violations
100
```

## checker\_get\_options

Use the `checker_get_options` command to return information on the types of checks Leda is configured to execute on the next run with the tool.

### Syntax

```
checker_get_options [-format language] [-block] [-chip] [-netlist]
[-sdc]
```

### Arguments

`-format` Returns the *language* for rules selected for checking. If you have rules that apply to both Verilog and VHDL selected for checking in your current configuration (very common, even if you are not checking a mixed-language design), this option returns both HDL names.

|                       |                                                                                                            |
|-----------------------|------------------------------------------------------------------------------------------------------------|
| <code>-block</code>   | Returns true (1) if block-level rule checking is enabled. It is enabled by default.                        |
| <code>-chip</code>    | Returns true (1) if chip-level rule checking is enabled. It is enabled by default.                         |
| <code>-netlist</code> | Returns true (1) if netlist checks are enabled. It is enabled by default.                                  |
| <code>-sdc</code>     | Returns true (1) if the Synopsys Design Constraint (SDC) checker is enabled. It is not enabled by default. |

### Example

The following example returns true (1) on the current project for chip-level checks, block-level checks, and netlist checks because they are all enabled by default:

```
leda> checker_get_options -chip -block -netlist
1 1 1
```

## checker\_set\_design\_constraints

Use the `checker_set_design_constraints` command to set information that the Checker uses for your next run with the tool. These options control the behavior of the tool and can affect runtime performance and the features that are enabled. Many of these options do the same things in Tcl shell mode as they do in batch mode (see [“Common Command-Line Options and Switches”](#) on page 148).

### Syntax

```
checker_set_design_constraints [-nowarning] \
 [-top {library_name top_unit_name} [-case_analysis_file
 file_name] [-case_analysis file_name] [-nohierdump] [-maxhierdump
 [limit]] [-forcehierdump] [-clockdump]
 [-noclockdump] [-test_clk_rising {clock_name}] \
 [-test_clk_falling {clock_name}] [-test_async {reset_name}] \
 [-test_async_inverted {reset_name}] [-max_violations number] \
 [-nomaxviolations]
```

### Arguments

- nowarning            Use the `-nowarning` switch to suppresses warning messages generated during compilation and elaboration.
- top                 Use the `-top` option to specify the *top\_unit\_name* or *library\_name* of your design hierarchy. This is required in order to check for chip-level rule violations.
- case\_analysis\_file   Use the `-case_analysis_file` option to point Leda to an ASCII text file containing `set_case_analysis` commands that specify constant values for primary inputs or internal signals. For more information, see [“Propagating Constants”](#) on page 96.
- nohierdump         Use the `-nohierdump` switch to turn off generation of the hierarchy browsing database. Optionally, also specify a *limit* for the hierarchy dump instead of a full disable.
- maxhierdump         Use the `-maxhierdump` switch to disable generation of the hierarchy browsing database beyond *limit* levels off hierarchy. This speeds up tool performance, but limits the hierarchy browser in the GUI after a Checker run.
- forcehierdump       Enable full generation of the hierarchy browsing database.
- clockdump           In `-full_log` mode, use the `-clockdump` switch to enable use of the Clock and Reset Tree browsers when you load a log file into the Error Viewer after checking your design. Note that

this switch can slow Leda's performance when checking large netlists. See [“Using the Clock and Reset Tree Browsers” on page 126](#).

- `-noclockdump` Use the `-noclockdump` switch to disable generation of the clock and reset browser.
- `-test_clk_rising` Use the `-test_clk_rising` option to specify test clock signal *clock\_name* and specify that the first edge in this clock's cycle is the rising edge. With RTLDR©, this corresponds to the following command:

```
create_test_clk CLK -w{N1 N1+N2}
```

In Leda's DFT checks, no delays are taken into account. Leda always assumes that the test clock period is 100 ns and the strobe point occurs at 95 ns (default RTLDR© values). Leda also assumes that all test clock events occur before this strobe point.

**Note:** The `-test_clk_rising` option expects the *CLK* argument to be a primary I/O name, not a hierarchical name including the top module name. Internal signals are not allowed.

- `-test_clk_falling` Use the `-test_clk_falling` option to specify test clock signal *clock\_name* and specify that the first edge in this clock's cycle is the falling edge. With RTLDR©, this corresponds to the following command:

```
create_test_clk CLK -w{N1 N1-N2}
```

In Leda's DFT checks, no delays are taken into account. Leda always assumes that the test clock period is 100 ns and the strobe point occurs at 95 ns (default RTLDR© values). Leda also assumes that all test clock events occur before this strobe point.

**Note:** The `-test_clk_falling` option expects the *CLK* argument to be a primary I/O name, not a hierarchical name including the top module name. Internal signals are not allowed.

- `-test_async` Use the `-test_async` option to specify the test reset signal *reset\_name* and indicate that it is active on “1” and has a hold value of “0” during the scan shift phase. With RTLDR©, this corresponds to the following command:

```
set_signal_type test_async RST
```

- `-test_async_inverted` Use the `-test_async_inverted` option to specify the test reset signal *reset\_name* and specify that it is active on “0” and has a hold value of “1” during the scan shift phase. With RTLDR©, this corresponds to the following command:
- ```
set_signal_type test_async_inverted RST
```
- `-max_violations` Use the `-maxviolations` option to set the maximum number of violations per selected rule that Leda flags. The default is 100.
- `-nomaxviolations` Use the `-nomaxviolations` switch to remove any limitations on the number of violations per selected rule that Leda flags. The default is 100.

Example

The following example sets the maximum number of errors that Leda flags for any one rule selected for checking to 50. This command does not return a value when it completes successfully.

```
leda> checker_set_design_constraints -max_violations 50
```

checker_set_options

Use the `checker_set_options` command to set up the kinds of checks you want to run in the tool. You must have the appropriate rules selected for checking in your configuration (block, chip, netlist) in order to get any results. For example, to run the netlist checker you must have some netlist checker rules selected.

Syntax

```
checker_set_options [-format language] [-block] [-chip] [-netlist] [-sdc]
```

Arguments

- `-format` Selects the *language* for rule checking. Valid values include `verilog` and `vhdl`.
- `-block` All checks are enabled by default. Use the `-block` switch if you want to run only block-level checks. These are unit-wide checks (for example, language-based checks).
- `-chip` All checks are enabled by default. Use the `-chip` switch if you want to run only chip-level checks. These are unit-wide checks that use hardware inference at the RTL level (for example, all flip-flops in the design must be active on the rising edge).

- netlist All checks are enabled by default. Use the -netlist switch if you want to run only netlist checks. These checks are run on the gate-level design netlist.
- sdc All checks are enabled by default. Use the -sdc switch if you want to run only SDC checks. These are checks on Synopsys Design Constraint files (see [“Using the SDC Checker” on page 133](#)).

Example

The following example sets the format to Verilog for the next run with the Checker. This command does not return a value when it completes successfully.

```
leda> checker_set_options -format verilog
```

current_design

Use the `current_design` command to define the design.

Syntax

```
current_design [-work library_name] [-nowarning] \
  [-top {[library_name] [top_unit_name]} [-case_analysis_file
file_name] [-nohierdump] [-maxhierdump [limit]] [-forcehierdump] \
  [-clockdump] [-noclockdump]
```

Arguments

- nowarning Use the `-nowarning` switch to suppresses warning messages generated during compilation and elaboration.
- top Use the `-top` option to specify the `top_unit_name` or `library_name` of your design hierarchy. This is required in order to check for chip-level rule violations.
- case_analysis_file Use the `-case_analysis_file` option to point Leda to an ASCII text file containing `set_case_analysis` commands that specify constant values for primary inputs or internal signals. For more information, see [“Propagating Constants” on page 96](#).
- nohierdump Use the `-nohierdump` switch to turn off generation of the hierarchy browsing database. Optionally, also specify a `limit` for the hierarchy dump instead of a full disable.
- maxhierdump Use the `-maxhierdump` switch to disable generation of the hierarchy browsing database beyond `limit` levels of hierarchy. This speeds up tool performance, but limits the hierarchy browser in the GUI after a Checker run.
- forcehierdump Enable full generation of the hierarchy browsing database.
- clockdump Use the `-clockdump` switch to enable use of the Clock and Reset Tree browsers when you load a log file into the Error Viewer after checking your design. Note that this switch can slow Leda’s performance when checking large netlists. See [“Using the Clock and Reset Tree Browsers” on page 126](#).
- noclockdump Use the `-noclockdump` switch to disable generation of the clock and reset browser.
- work If there is more than one working library defined for the current project, you must specify a working `library_name`; otherwise this is optional.

Example

The following example specifies that top is the top-level unit and elaborates the design, but does not run any checks. This command does not return a value when it completes successfully.

```
leda> current_design top
```

elaborate

Use the elaborate command to elaborate the design. This is required before you can use any of the DQL netlist query commands documented in the [Leda Tcl Interface Guide](#). Before you run the elaborate command, run the Leda Checker at least once on your design. This command does the same thing as the [propagate](#) command.

Syntax

```
elaborate [-work library_name] [-netlist] [-nowarning] \  
  [-top { [library_name] [top_unit_name] } [-case_analysis_file  
  file_name] [-nohierdump] [-maxhierdump limit] [-forcehierdump] \  
  [-clockdump] [-noclockdump]
```

Arguments

- | | |
|---------------------|--|
| -netlist | Use the -netlist switch to enable checking of design netlist rules. |
| -nowarning | Use the -nowarning switch to suppresses warning messages generated during compilation and elaboration. |
| -top | Use the -top option to specify the <i>top_unit_name</i> or <i>library_name</i> of your design hierarchy. This is required in order to check for chip-level rule violations. |
| -case_analysis_file | Use the -case_analysis_file option to point Leda to an ASCII text file containing set_case_analysis commands that specify constant values for primary inputs or internal signals. For more information, see “Propagating Constants” on page 96 . |
| -nohierdump | Use the -nohierdump switch to turn off generation of the hierarchy browsing database. Optionally, also specify a <i>limit</i> for the hierarchy dump instead of a full disable. |
| -maxhierdump | Use the -maxhierdump switch to disable generation of the hierarchy browsing database beyond <i>limit</i> levels off hierarchy. This speeds up tool performance, but limits the hierarchy browser in the GUI after a Checker run. |
| -forcehierdump | Enable full generation of the hierarchy browsing database. |

- `-clockdump` Use the `-clockdump` switch to enable use of the Clock and Reset Tree browsers when you load a log file into the Error Viewer after checking your design. Note that this switch can slow Leda's performance when checking large netlists. See [“Using the Clock and Reset Tree Browsers” on page 126](#).
- `-noclockdump` Use the `-noclockdump` switch to disable generation of the clock and reset browser.
- `-work` Specify a working *library_name*.

Example

The following example specifies that `top` is the top-level unit and elaborates the design, but does not run any checks:

```
leda> elaborate -top top  
Executing elaboration of top unit top ...  
Elaboration of top unit top completed.  
Dumping the design hierarchy...  
Design hierarchy dump completed.
```

link

Use the link command to elaborate the design. This is required before you can use any of the DQL netlist query commands documented in the *Leda Tcl Interface Guide*. Before you run the link command, run the Leda Checker at least once on your design. This command does the same thing as the [propagate](#) command.

Syntax

```
link [-work library_name] [-netlist] [-nowarning] \  
    [-top { [library_name] [top_unit_name] } [-case_analysis_file  
    file_name] [-nohierdump] [-maxhierdump limit] [-forcehierdump] \  
    [-clockdump] [-noclockdump]
```

Arguments

-netlist	Use the -netlist switch to enable checking of design netlist rules.
-nowarning	Use the -nowarning switch to suppresses warning messages generated during compilation and elaboration.
-top	Use the -top option to specify the <i>top_unit_name</i> or <i>library_name</i> of your design hierarchy. This is required in order to check for chip-level rule violations.
-case_analysis_file	Use the -case_analysis_file option to point Leda to an ASCII text file containing set_case_analysis commands that specify constant values for primary inputs or internal signals. For more information, see “Propagating Constants” on page 96 .
-nohierdump	Use the -nohierdump switch to turn off generation of the hierarchy browsing database. Optionally, also specify a <i>limit</i> for the hierarchy dump instead of a full disable.
-maxhierdump	Use the -maxhierdump switch to disable generation of the hierarchy browsing database beyond <i>limit</i> levels of hierarchy. This speeds up tool performance, but limits the hierarchy browser in the GUI after a Checker run.
-forcehierdump	Enable full generation of the hierarchy browsing database.
-clockdump	Use the -clockdump switch to enable use of the Clock and Reset Tree browsers when you load a log file into the Error Viewer after checking your design. Note that this switch can slow Leda’s performance when checking large netlists. See “Using the Clock and Reset Tree Browsers” on page 126 .

- noclockdump Use the -noclockdump switch to disable generation of the clock and reset browser.
- work Specify a working *library_name*.

Example

The following example specifies that top is the top-level unit and elaborates the design, but does not run any checks:

```
leda> link -top top
Executing elaboration of top unit top ...
Elaboration of top unit top completed.
Dumping the design hierarchy...
Design hierarchy dump completed.
```

propagate

Use the propagate command to propagate values set with set_case_analysis commands in your SDC file. Before you propagate values, use the [propagate](#) command to apply them to the elaborated database.

Syntax

```
propagate -case_analysis
```

Arguments

- case_analysis Use the -case_analysis switch to propagate values set in set_case_analysis commands in the SDC file.

Example

The following example does not return any values when it completes successfully:

```
leda> set_case_analysis 1 test_in #sets a value to a port
leda> read_constraints my.sdc #reads some SDC constraints
leda> sdc_apply -case_analysis \
#sets values to ports/pins as indicated by any set_case_analysis
commands in my.sdc
leda> propagate #propagates the values set on pins through the design
```

read_constraints

Use the `read_constraints` command to read a Synopsys Design Constraint (SDC) file into an internal database used by the SDC Checker. This command does not source the SDC file. Note that Leda parses application-specific SDC commands (e.g., for PrimeTime and Design Compiler) but does no further processing for them. Read in SDC files in top-down order (that is, read files containing declarations before reading files that use those declarations). For information on how to use the SDC Checker, see [“Using the SDC Checker” on page 133](#). For a detailed list of the supported SDC file commands, see [“Supported SDC File Tcl Commands” on page 137](#).

Syntax

```
read_constraints file_name [-block instance_name sdc_block_file_name]\
[-for_equivalency]
```

Arguments

<code>-block</code>	Use the <code>-block</code> switch to specify a block-level constraint file of a design to check.
<i>instance_name</i>	Specify the instance name that is instantiated inside the top module.
<i>file_name</i>	Specify the SDC <i>file_name</i> of the top module of the design to check.
<code>-for_equivalency</code>	Use the <code>-for_equivalency</code> switch to access the equivalency checks

Example

The following example illustrates the usage of above commands:

```
leda> read_verilog test.v
leda> elaborate -top TOP #elaborating the top module "TOP"
leda> read_constraints top.sdc #reading the top level constraint file
leda> read_constraints -block B1 B1.sdc #reading the constraint file of
                                #block B1
leda> read_constraints -block B2 B2.sdc #reading the constraint file of
                                #block B2
leda> check -sdc -config
```

The following example illustrates the usage of `-for_equivalency` switch:

```
leda> read_verilog -f files.list
leda> elaborate -top TOP #elaborating the top module "TOP"
leda> read_constraints SDC1.tcl
leda> read_constraints -for_equivalency SDC2.tcl
leda> check -sdc -config sdc_eqv_rules.tcl
```

For more information, see the [Leda Constraints Rules Guide](#).

read_files

Use the `read_files` command to compile Verilog or VHDL source files. This command compiles the files, but does not check them. This command is equivalent to using `leda -c` in batch mode. When you use this command, Leda analyzes the code for compatibility with Verilog or VHDL syntax and semantics.

Syntax

```
read_files file_names [-project project_name] [-f file_name] \
  [-files file_name] [-netlist_reader file_name] [-lang LANG] \
  [-format language] [-mk] [-mkk] [-max_violations number] \
  [-nowarning] [-work library_name] [-translate_directive] \
  [-severity level] [-v library_file] [-y library_dir] \
  [+checklib+<libname>] [+define+macro [=val]] [+libext{+.string}] \
  [+incdir {+directory}] [+v2k] [+sv] [-sverilog] \
  [-nochecklib library_name]
```

Arguments

<i>file_names</i>	Specify the Verilog or VHDL <i>file_names</i> to compile.
<code>-project</code>	Specify the <i>project_name</i> to associate the files with.
<code>-f</code>	Verilog only. Use the <code>-f</code> option to specify a command file (<i>file_name</i>) that lists Verilog files and any other options that you want to specify.
<code>-files</code>	VHDL only. Use the <code>-files</code> option to specify that all VHDL files to be analyzed and checked are listed in the text file <i>file_name</i> . If you use this option in conjunction with the <code>-project</code> option, a <code>#Files</code> or <code>#Dirs</code> clause in the file indicated by the <i>file_name</i> argument must contain the library name.
<code>-lang</code>	Use the <code>-lang</code> option to select the <i>LANG</i> mode to use when analyzing code. This option can take one of the following values: <ul style="list-style-type: none"> •87—analyzed using VHDL 87 syntax and semantics. •87e—analyzed using VHDL 87 syntax and semantics, with some semantic exceptions. •93—analyzed using VHDL 93 syntax and semantics. •93e—analyzed using VHDL 93 syntax and semantics, with some semantic exceptions. This is the default for VHDL.

- 95—analyzed using syntax and semantics specified in the Verilog LRM.
 - 95e—analyzed as Verilog 95, but with some commonly used semantic exceptions. Emulates analyzers that do not conform to the Verilog LRM. This is the default for Verilog.
- format Specify the *language*. Legal values include vhdl and verilog. Default is all.
- mk VHDL only. Use the -mk switch to make Leda automatically deduce the compilation order for VHDL source files.
- mkk VHDL only. Same as -mk, but continues if there is a syntax error.
- max_violations Use the -max_violations option to set the maximum *number* of violations per selected rule that Leda flags. The default is 100.
- netlist_reader Use the -netlist_reader option to invoke the netlist reader.
- nowarning Use the -nowarning switch to suppresses warning messages generated during compilation and elaboration.
- work Use the -work option to specify the name of the working *library* into which all files will be compiled. The default working library is LEDA_WORK. You specify the physical location of the logical library in a file called plibs (see [“Using plibs to Set Library Logical/Physical Mapping”](#) on page 146). If you do not specify the -work option, Leda analyzes the plibs file to see if there is a physical library mapped to the logical name WORK. If not, the library .leda_work is used. If the plibs file contains no physical location for this library, Leda creates it locally.
- translate_directive Use the -translate_directive switch to make Leda ignore code between Synopsys synthesis off/on and translate off/on compiler directives.

- severity Specify the severity *level* for which you want to see messages from the analyzer. You get the selected severity level and above. Legal values include note, warning, error, and fatal.
- v Verilog only. Use the -v option to specify a *library_file*. The Checker scans each library file for module definitions and tries to resolve all unresolved module instances in the Verilog source files.
- Note:** This option works just like the VCS -v option, except that Leda does not include modules coming from files specified after -v unless you also use the +checklib option.
- y Verilog only. Use the -y option to specify a *library_directory* that contains Verilog source files. The Checker scans the files in each library directory for module declarations and tries to resolve all unresolved module instances in the Verilog source files. This option work for files containing more than one module.
- +checklib+<libname> Verilog only. If *libname* refers to a directory specified with -y, Leda includes all modules found in that directory.
- +define+macro [=val] Verilog only. Use the +define option to define the *macro* and assign *val* to it.
- +libext{+.string} Verilog only. Use the +libext option to specify file extensions for files in library directories (see option y). You can only use one libext clause on the command line. The default file extensions for the -y option are .v and .V.
- +incdir {+directory} Verilog only. Use the +incdir option to specify the directories to be searched for included files.
- +v2k Verilog only. Use the +v2k switch to make Leda parse and check language compliance for supported Verilog 2001 constructs. For information on current supported constructs, see [“Verilog 2001 Support” on page 65](#).
- Note:** This is the same switch used with the Synopsys VCS simulator.
- +sv Verilog only. Use the +sv switch to make Leda parse and check language compliance for SystemVerilog.

- `-sverilog` Use the `-sverilog` switch to make Leda parse and check language compliance for SystemVerilog. This works the same way as `-sverilog`, but is present for compatibility with the VCS command line.
- `-nochecklib` VHDL only. Specify VHDL resource libraries that you don't want Leda to check for errors.

Example

The following example reads the `eightadd.v` Verilog source file. This command does not return a value when it completes successfully.

```
leda> read_files /u/me/Verilog/eightadd.v -format verilog
```

read_sverilog

Use the `read_sverilog` command to compile SystemVerilog source files. This command compiles the files, but does not check them. When you use this command, Leda analyzes the code for compatibility with SystemVerilog syntax and semantics.

Syntax

```
read_sverilog file_names [-project project_name] [-f file_name] \
  [-netlist_reader file_name] [-u] [-work library_name] \
  [-v library_file] [-nowarning] [-translate_directive] \
  [-max_violations number] [-severity level] [-y library_directory] \
  [+checklib+<libname>] [+define+macro [=val]] [+libext{+.string}] \
  [+incdir {+directory}] [-nowarning]
```

Arguments

<i>file_names</i>	Specify the <i>file_names</i> of SystemVerilog files to compile.
-project	Specify the <i>project_name</i> to associate the files with.
-f	Use the -f option to specify a command file (<i>file_name</i>) that can list SystemVerilog files and any other options that you want to specify.
-u	Do not use case to distinguish identifier names.
-max_violations	Use the -max_violations option to set the maximum <i>number</i> of violations per selected rule that Leda flags. The default is 100.
-severity	Specify the severity <i>level</i> for which you want to see messages from the analyzer. You get the selected severity level and above. Legal values include note, warning, error, and fatal.
-netlist_reader	Use the -netlist_reader option to invoke the netlist reader.
-nowarning	Use the -nowarning switch to suppresses warning messages generated during compilation and elaboration.
-work	Use the -work option to specify the name of the <i>library_name</i> into which all files will be compiled. The default working library is LEDA_WORK. You specify the physical location of the logical library in a file called plibs (see “Using plibs to Set Library Logical/Physical Mapping” on page 146). If you do not specify the -work option, Leda analyzes the plibs file to see if there is a physical library mapped to the logical name

WORK. If not, the library `.leda_work` is used. If the `plibs` file contains no physical location for this library, Leda creates it locally.

- translate_directive Use the `-translate_directive` switch to make Leda ignore code between Synopsys synthesis off/on and translate off/on compiler directives.
- v Use the `-v` option to specify a *library_file*. The Checker scans the *library_file* for module definitions and tries to resolve all unresolved module instances in the SystemVerilog source files. Note: This option works just like the VCS `-v` option, except that Leda does not include modules coming from files specified after `-v` unless you also use the `+checklib` option.
- y Use the `-y` option to specify a *library_directory* that contains SystemVerilog source files. The Checker scans the files in each library directory for module declarations and tries to resolve all unresolved module instances in the SystemVerilog source files. This option work for files containing more than one module.
- +checklib+<libname> If *libname* refers to a directory specified with `-y`, Leda includes all modules found in that directory.
- +define+macro [=val] Use the `+define` option to define the *macro* and assign *val* to it.
- +libext{+.string} Use the `+libext` option to specify file extensions for files in library directories (see option `y`). You can only use one `libext` clause on the command line. The default file extensions for the `-y` option are `.v` and `.V`.
- +incdir{+directory} Use the `+incdir` option to specify *directories* to search for included files.

Note: This is the same switch used with the Synopsys VCS simulator.
- nowarning Use the `-nowarning` switch to suppresses warning messages generated during compilation and elaboration.

Example

The following example reads a SystemVerilog source file and puts the results of the analysis in the WORK working directory.

```
leda> read_sverilog -work WORK /u/me/Verilog/eightadd.sv
Reading file eightadd.sv (no check)
```

read_verilog

Use the `read_verilog` command to compile Verilog source files. This command compiles the files, but does not check them. When you use this command, Leda analyzes the code for compatibility with Verilog syntax and semantics.

Syntax

```
read_verilog file_names [-project project_name] [-f file_name] \
  [-netlist_reader file_name] [-u] [-lang LANG] [-work library_name] \
  [-v library_file] [-nowarning] [-translate_directive] \
  [-max_violations number] [-severity level] [-y library_directory] \
  [+checklib+<libname>] [+define+macro [=val]] [+libext{+.string}] \
  [+incdir {+directory}] [+v2k] [+sv] [-sverilog] \
  [-nowarning]
```

Arguments

<i>file_names</i>	Specify the <i>file_names</i> of Verilog files to compile.
-project	Specify the <i>project_name</i> to associate the files with.
-f	Use the -f option to specify a command file (<i>file_name</i>) that can list Verilog files and any other options that you want to specify.
-u	Do not use case to distinguish identifier names.
-lang	Use the -lang option to select the <i>LANG</i> mode to use when analyzing code. This option can take one of the following values: <ul style="list-style-type: none"> •95—analyzed using syntax and semantics specified in the Verilog LRM. •95e—analyzed as Verilog 95, but with some commonly used semantic exceptions. Emulates analyzers that do not conform to the Verilog LRM. This is the default.
-max_violations	Use the -max_violations option to set the maximum <i>number</i> of violations per selected rule that Leda flags. The default is 100.
-severity	Specify the severity <i>level</i> for which you want to see messages from the analyzer. You get the selected severity level and above. Legal values include note, warning, error, and fatal.
-netlist_reader	Use the -netlist_reader option to invoke the netlist reader.

- nowarning** Use the `-nowarning` switch to suppresses warning messages generated during compilation and elaboration.
- work** Use the `-work` option to specify the name of the *library_name* into which all files will be compiled. The default working library is LEDA_WORK. You specify the physical location of the logical library in a file called plibs (see [“Using plibs to Set Library Logical/Physical Mapping” on page 146](#)). If you do not specify the `-work` option, Leda analyzes the plibs file to see if there is a physical library mapped to the logical name WORK. If not, the library `.leda_work` is used. If the plibs file contains no physical location for this library, Leda creates it locally.
- translate_directive** Use the `-translate_directive` switch to make Leda ignore code between Synopsys synthesis off/on and translate off/on compiler directives.
- v** Use the `-v` option to specify a *library_file*. The Checker scans the *library_file* for module definitions and tries to resolve all unresolved module instances in the Verilog source files. Note: This option works just like the VCS `-v` option, except that Leda does not include modules coming from files specified after `-v` unless you also use the `+checklib` option.
- y** Use the `-y` option to specify a *library_directory* that contains Verilog source files. The Checker scans the files in each library directory for module declarations and tries to resolve all unresolved module instances in the Verilog source files. This option work for files containing more than one module.
- +checklib+<libname>**If *libname* refers to a directory specified with `-y`, Leda includes all modules found in that directory.
- +define+macro [=val]**Use the `+define` option to define the *macro* and assign *val* to it.
- +libext{+.string}** Use the `+libext` option to specify file extensions for files in library directories (see option `y`). You can only use one `libext` clause on the command line. The default file extensions for the `-y` option are `.v` and `.V`.
- +incdir{+directory}** Use the `+incdir` option to specify *directories* to search for included files.

- `+v2k` Use the `+v2k` switch to make Leda parse and check language compliance for supported Verilog 2001 constructs (see [“Verilog 2001 Support” on page 65](#)).
- Note:* This is the same switch used with the Synopsys VCS simulator.
- `+sv` Use the `+sv` switch to make Leda parse and check language compliance for SystemVerilog (see [“SystemVerilog Support” on page 65](#)).
- `-sverilog` Use the `-sverilog` switch to make Leda parse and check language compliance for SystemVerilog. This works the same way as `-sverilog`, but is present for compatibility with the VCS command line.
- `-nowarning` Use the `-nowarning` switch to suppresses warning messages generated during compilation and elaboration.

Example

The following example reads a Verilog source file and puts the results of the analysis in the WORK working directory.

```
leda> read_verilog -work WORK /u/me/Verilog/eightadd.v
Reading file eightadd.v (no check)
```

read_vhdl

Use the `read_vhdl` command to compile VHDL source files. This command compiles the files, but does not check them. When you use this command, Leda analyzes the code for compatibility with VHDL syntax and semantics.

Syntax

```
read_vhdl file_names [-project project_name] [-lang LANG] [-mk] [-mkk] \
  [-max_violations number] [-nowarning] [-translate_directive] \
  [-work library_name] [-severity level]
```

Arguments

<i>file_names</i>	Specify the VHDL <i>file_names</i> to compile.
<code>-project</code>	Specify the <i>project_name</i> (a container for the VHDL source files that you want to check).
<code>-lang</code>	Use the <code>-lang</code> option to select the <i>LANG</i> mode to use when analyzing code. The default is VHDL 93. This option can take one of the following values: <ul style="list-style-type: none"> •87—analyzed using VHDL 87 syntax and semantics. •87e—analyzed using VHDL 87 syntax and semantics, with some semantic exceptions. •93—analyzed using VHDL 93 syntax and semantics. •93e—analyzed using VHDL 93 syntax and semantics, with some semantic exceptions. This is the default.
<code>-mk</code>	Use the <code>-mk</code> switch to make Leda automatically deduce the compilation order for your VHDL source files.
<code>-mkk</code>	Same as <code>-mk</code> switch, except that compilation continues even if there is a syntax error.
<code>-max_violations</code>	Use the <code>max_violations</code> option to set the maximum <i>number</i> of violations Leda should print per selected rule in your configuration. The default is 100.
<code>-nowarning</code>	Use the <code>-nowarning</code> switch to suppress compiler warnings.
<code>-translate_directive</code>	Use the <code>-translate_directive</code> switch to make Leda ignore code between Synopsys synthesis off/on and translate off/on compiler directives.

- work** Use the `-work` option to specify the name of the *library_name* into which all files will be analyzed. This option is ignored if you do not specify any files on the command line. You specify the physical location of the logical library in a file called `plibs` (see [“Using plibs to Set Library Logical/Physical Mapping” on page 146](#)). If you do not specify the `-work` option, Leda analyzes the `plibs` file to see if there is a physical library mapped to the logical name `WORK`. If not, the library `.leda_work` is used. If the `plibs` file contains no physical location for this library, Leda creates it locally.
- severity** Specify the minimum severity *level* for which compiler messages are displayed.

Example

The following example analyzes the `entity.vhd` VHDL source file according to the VHDL 93 standard, with some semantic exceptions:

```
leda> read_vhdl -work WORK /u/me/VHDL/entity.vhd -lang 93e
Reading file entity.vhd (no check)
```

report

Use the `report` command after running Leda to get error reports on STDOUT for individual rules or all rules selected for checking in that run with the tool. If you don't select one of the options, Leda prints all error reports by default.

Syntax

```
report [all] [rule_name]
```

Arguments

- all** Use the `all` argument to get error reports for all rules violated in the last check. This is the default.
- rule_name** Use the `rule_name` argument to get a report on one particular rule violated in the last check.

Example

The following example generates a report for rule `B_3417`:

```
leda> report B_3417
11: always @(negedge clk) q = d;
      ^

reg.v:11: STATEM> [ERROR] B_3417: Use non-blocking assignments in
sequential block
```

run

Use the run command to run the tool on the current project (HDL source files) using the specified options. You must have the appropriate rules selected for checking in your configuration (block, chip, netlist) in order to get any results. For example, to run the netlist checker you must have some netlist checker rules selected. If you execute the run command without specifying any options, Leda runs the Checker using the last values for any options already set in that session. This command works just like the [propagate](#) command.

Syntax

```
run [-format language] [-work library_name] [-block] [-chip] \
  [-netlist] [-nocheck] [-propagate] [-config file_name] \
  [-policy policy_name] [-ruleset ruleset_name] [-rule rule_name] \
  [-nowarning] [-top {[library_name] top_unit_name}] \
  [-noelaboration] [-case_analysis_file file_name] [-nohierdump] \
  [-clockdump] [-test_clk_rising clock_name] \
  [-test_clk_falling clock_name] [-test_async reset_name] \
  [-test_async_inverted reset_name] [-max_violations number] \
  [-nohierdump limit] [+max_case+<val>] [+max_casexz+<val>] \
  [-full_inf]
```

Arguments

- format Selects the *language* for rule checking. Valid values include verilog and vhdl. If you don't specify either option, Leda checks selected rules for both Verilog and VHDL by default.
- work Specify a working *library_name*.
- block All checks are enabled by default starting with version 4.1. Use the -block switch if you want to run only block-level checks. These are unit-wide checks that are like traditional lint checks (for example, language-based checks).
- chip All checks are enabled by default starting with version 4.1. Use the -chip switch if you want to run only chip-level checks. These are unit-wide checks that use hardware inference at the RTL level (for example, all flip-flops in the design must be active on the rising edge).
- full_inf This option will enable the user to display the violation summary and the deactivated rules in the .inf file.
- netlist All checks are enabled by default starting with version 4.1. Use the -netlist switch if you want to run only netlist checks. These checks are run on the gate-level design netlist.

- nocheck Elaborate the design but don't run any rule checks.
- propagate Enable constant propagation. See the `-case_analysis` option, which you use to point to a file that contains `set_case_analysis` commands that make constant propagation work.
- config Specify the full path to the configuration file that you want to use for this run with the tool.
- policy Specify a *policy_name* that you want to check. If you use this option, Leda ignores rule selections specified in your current configuration.
- ruleset Specify a *ruleset_name* that you want to check. If you use this option, Leda ignores rule selections specified in your current configuration.
- rule Specify a *rule_label* for a rule that you want to check. If you use this option, Leda ignores rule selections specified in your current configuration.
- nowarning Optional, and only relevant for chip-level checks. Use the `-nowarning` switch to suppresses warning messages generated during compilation and elaboration.
- top Use the `-top` option to specify the *top_unit_name* or *library_name* for your design hierarchy. This is required in order to check for chip-level rule violations.
- noelaboration Do not force re-elaboration if it is not necessary.
- case_analysis_file Use the `-case_analysis_file` option to point Leda to an ASCII text file containing `set_case_analysis` commands that specify constant values for primary inputs or internal signals used in constant propagation. For more information, see [“Propagating Constants” on page 96](#).
- nohierdump Use the `-nohierdump` switch to turn off generation of the hierarchy browsing database. Optionally, also specify a *limit* for the hierarchy dump instead of a full disable. This speeds up tool performance, but disables the hierarchy browser in the GUI after a Checker run.
- clockdump Use the `-clockdump` switch to enable use of the Clock and Reset Tree browsers when you load a log file into the Error Viewer after checking your design. Note that this switch can slow Leda's performance when checking large netlists. See [“Using the Clock and Reset Tree Browsers” on page 126](#).

- `-max_violations` Use the `-max_violations` option to set the maximum number of violations per selected rule that Leda flags. The default is 100.
- `+max_case+<val>` Use the `+max_case` option to specify the maximum width of a case expression in a case statement. The default value is 8.
- `+max_casexz+<val>` Use the `+max_casexz` option to specify the maximum width of a case expression in a casex/casez statement. The default value is 8.

Example

The following example runs all rules selected in the current configuration:

```
leda> run -top top
Executing elaboration of top unit top ...
Elaboration of top unit top completed.
Dumping the design hierarchy...
Design hierarchy dump completed.
Executing chip-level checks on design top ...
Chip-level checks on design top completed.
```

sdc_apply

Use the `sdc_apply` command to apply `set_case_analysis` data read from SDC files to the elaborated database. After you apply the data, use the [propagate](#) command to propagate the values,

Syntax

```
sdc_apply -case_analysis
```

Arguments

- `-case_analysis` Use the `-case_analysis` switch to apply values set with `set_case_analysis` commands in the SDC file.

Example

The following example does not return any values when it completes successfully:

```
leda> set_case_analysis 1 test_in #sets a value to a port
leda> read_constraints my.sdc #reads some SDC constraints
leda> sdc_apply -case_analysis \
#sets values to ports/pins as indicated by any set_case_analysis
commands in my.sdc
leda> propagate #propagates the values set on pins through the design
```

set_case_analysis

Use the `set_case_analysis` command to set the value of a constant that you want Leda to propagate through the design. For information on constant propagation, see [“Propagating Constants” on page 96](#).

Syntax

```
set_case_analysis value port_or_pin_list
```

Arguments

<i>value</i>	Specify the constant <i>value</i> . Legal values are ZERO, ONE, 0 or 1.
<i>port_or_pin_list</i>	List the <i>port_or_pin_list</i> that you want held at the specified value. Legal values include the signal name (b), hierarchical name (top.b), or a list {a c top.b}.

Example

The following example sets the P1 signal to 0 and holds it at that value for the Checker run.

```
leda> set_case_analysis 0 P1
```

The following example sets CNT (bus signal) to 0 and holds it at that value for the checker run.

```
leda> set_case_analysis 0 CNT(0)
leda> set_case_analysis 0 CNT(1)
leda> set_case_analysis 0 CNT(2)
leda> set_case_analysis 0 CNT(3)
```


verify

Use the `verify` command to run the tool on the current project (HDL source files) using the specified options. You must have the appropriate rules selected for checking in your configuration (block, chip, netlist) in order to get any results. For example, to run the netlist checker you must have some netlist checker rules selected. If you execute the `verify` command without specifying any options, Leda runs the Checker using the last values for any options already set in that session. This command does the same thing as the `propagate` command.

Syntax

```
verify [-format language] [-work library_name] [-block] [-chip] \
      [-netlist] [-nocheck] [-propagate] [-config file_name] \
      [-policy policy_name] [-ruleset ruleset_name] [-rule rule_name] \
      [-nowarning] [-top {[library_name] top_unit_name}] \
      [-noelaboration] [-case_analysis_file file_name] [-nohierdump] \
      [-clockdump] [-test_clk_rising clock_name] \
      [-test_clk_falling clock_name] [-test_async reset_name] \
      [-test_async_inverted reset_name] [-max_violations number] \
      [-nohierdump limit]
```

Arguments

- format Selects the *language* for rule checking. Valid values include `verilog` and `vhdl`. If you don't specify either option, Leda checks selected rules for both Verilog and VHDL by default.
- work Specify a working *library_name*.
- block All checks are enabled by default starting with version 4.1. Use the `-block` switch if you want to run only block-level checks. These are unit-wide checks that are like traditional lint checks (for example, language-based checks).
- chip All checks are enabled by default starting with version 4.1. Use the `-chip` switch if you want to run only chip-level checks. These are unit-wide checks that use hardware inference at the RTL level (for example, all flip-flops in the design must be active on the rising edge).
- netlist All checks are enabled by default starting with version 4.1. Use the `-netlist` switch if you want to run only chip-level checks. These checks are run on the gate-level design netlist.
- nocheck Elaborate the design but don't run any rule checks.

- propagate Enable constant propagation. See the `-case_analysis` option, which you use to point to a file that contains `set_case_analysis` commands that make constant propagation work.
- config Specify the full path to the configuration file that you want to use for this run with the tool.
- policy Specify a *policy_name* that you want to check. If you use this option, Leda ignores rule selections specified in your current configuration.
- ruleset Specify a *ruleset_name* that you want to check. If you use this option, Leda ignores rule selections specified in your current configuration.
- rule Specify a *rule_label* for a rule that you want to check. If you use this option, Leda ignores rule selections specified in your current configuration.
- nowarning Optional, and only relevant for chip-level checks. Use the `-nowarning` switch to suppresses warning messages generated during compilation and elaboration.
- top Use the `-top` option to specify the *top_unit_name* or *library_name* for your design hierarchy. This is required in order to check for chip-level rule violations.
- noelaboration Do not force re-elaboration if it is not necessary.
- case_analysis_file Use the `-case_analysis_file` option to point Leda to an ASCII text file containing `set_case_analysis` commands that specify constant values for primary inputs or internal signals used in constant propagation. For more information, see [“Propagating Constants” on page 96](#).
- nohierdump Use the `-nohierdump` switch to turn off generation of the hierarchy browsing database. Optionally, also specify a *limit* for the hierarchy dump instead of a full disable. This speeds up tool performance, but disables the hierarchy browser in the GUI after a Checker run.
- clockdump Use the `-clockdump` switch to enable use of the Clock and Reset Tree browsers when you load a log file into the Error Viewer after checking your design. Note that this switch can slow Leda’s performance when checking large netlists. See [“Using the Clock and Reset Tree Browsers” on page 126](#).

`-max_violations` Use the `-max_violations` option to set the maximum number of violations per selected rule that Leda flags. The default is 100.

Example

The following example runs all rules selected in the current configuration. In this case, only a few chip-level rules were selected, so those are the checks Leda executed:

```
leda> verify -top top
Executing chip-level checks on design top ...
Chip-level checks on design top completed.
```

Generating Log Files in Tcl Mode

When you create a project in the Tcl mode, a directory `leda-logs` is created in the present working directory. Files `leda.log` and `leda.inf` are created in this directory.

Reserved Variables

Leda's shell uses the following global constants. You can read these constants but you cannot modify them.

- `CCI_register_constant("A_RESET")`
- `CCI_register_constant("A_SET")`
- `CCI_register_constant("ACTIVE_HIGH")`
- `CCI_register_constant("ACTIVE_LOW")`
- `CCI_register_constant("ADD")`
- `CCI_register_constant("ALWAYS_ON")`
- `CCI_register_constant("AND")`
- `CCI_register_constant("BLACKBOX")`
- `CCI_register_constant("BUFFER")`
- `CCI_register_constant("BUFFERED")`
- `CCI_register_constant("BUS")`
- `CCI_register_constant("CELL")`
- `CCI_register_constant("CHECK_RECONVERGENT_FANOUT")`
- `CCI_register_constant("CHECKLIB")`
- `CCI_register_constant("CLOCK")`

- CCI_register_constant("CLOCK_SIGNAL_INDEX")
- CCI_register_constant("CMD_FILE")
- CCI_register_constant("COMPLEX")
- CCI_register_constant("CONCISE_REPORT")
- CCI_register_constant("CONTROL")
- CCI_register_constant("COUNT_ALL_OCCURRENCES")
- CCI_register_constant("COUNT_LOGIC_LEVEL")
- CCI_register_constant("DATA")
- CCI_register_constant("DB_OBJECT")
- CCI_register_constant("DEFINITION")
- CCI_register_constant("DQ_STOP_AT_BOUNDARY")
- CCI_register_constant("DQ_TRACE_LOGIC_LEVEL")
- CCI_register_constant("ENABLE")
- CCI_register_constant("ENABLE_HIGH")
- CCI_register_constant("ENABLE_LOW")
- CCI_register_constant("EXCLUDED")
- CCI_register_constant("FILE")
- CCI_register_constant("filter1_data")
- CCI_register_constant("filter1_type")
- CCI_register_constant("filter2_data")
- CCI_register_constant("filter2_type")
- CCI_register_constant("filter3_data")
- CCI_register_constant("filter3_type")
- CCI_register_constant("filter4_data")
- CCI_register_constant("filter4_type")
- CCI_register_constant("FLIPFLOP")
- CCI_register_constant("CCI_register_constant("GATE_NETLIST"))
- CCI_register_constant("GIVE_ALL_PATHS")
- CCI_register_constant("GIVE_EDGE_INFO")

- CCI_register_constant("GND")
- CCI_register_constant("GO_THROUGH_SEQ")
- CCI_register_constant("I")
- CCI_register_constant("IGNORE_CONSTANT_SIGNALS")
- CCI_register_constant("IMPLICIT")
- CCI_register_constant("INCLUDE_FILE")
- CCI_register_constant("INSTANCE")
- CCI_register_constant("INV_TRIEN")
- CCI_register_constant("INV_TRIENB")
- CCI_register_constant("INVERTED")
- CCI_register_constant("INVERTER")
- CCI_register_constant("IO")
- CCI_register_constant("ISOLATION_CELL")
- CCI_register_constant("LATCH")
- CCI_register_constant("LEVEL_SHIFTER")
- CCI_register_constant("LIST_ARGUMENT")
- CCI_register_constant("LIST_TYPE")
- CCI_register_constant("LOOP")
- CCI_register_constant("MEMORY")
- CCI_register_constant("MULT")
- CCI_register_constant("MUX_N")
- CCI_register_constant("MUX21")
- CCI_register_constant("MUX41")
- CCI_register_constant("NAND")
- CCI_register_constant("NEGEDGE")
- CCI_register_constant("NET")
- CCI_register_constant("NON_INVERTED")
- CCI_register_constant("NOR")
- CCI_register_constant("NORMAL_REPORT")

- CCI_register_constant("NOTIFIER")
- CCI_register_constant("O")
- CCI_register_constant("OR")
- CCI_register_constant("PAD")
- CCI_register_constant("PI")
- CCI_register_constant("PIO")
- CCI_register_constant("PO")
- CCI_register_constant("PORT")
- CCI_register_constant("POSEDGE")
- CCI_register_constant("POSITIONAL_SIGNAL_INDEX")
- CCI_register_constant("POWER")
- CCI_register_constant("POWER_BLOCK")
- CCI_register_constant("POWER_REGION")
- CCI_register_constant("POWER_SWITCH")
- CCI_register_constant("PRIMITIVE")
- CCI_register_constant("PRIORITY_PIN")
- CCI_register_constant("Q")
- CCI_register_constant("QN")
- CCI_register_constant("S_RESET")
- CCI_register_constant("S_SET")
- CCI_register_constant("SCAN_CLOCK")
- CCI_register_constant("SCAN_ENABLE")
- CCI_register_constant("SCAN_IN")
- CCI_register_constant("SIGNAL")
- CCI_register_constant("STMT")
- CCI_register_constant("STOP_AT_ANY_SIGNAL")
- CCI_register_constant("STOP_AT_COMPLEX")
- CCI_register_constant("STOP_AT_PORT")
- CCI_register_constant("SUPPLY0")

- CCI_register_constant("SUPPLY1")
- CCI_register_constant("SYMBOL_SIGNAL_INDEX")
- CCI_register_constant("TOP_MODULE")
- CCI_register_constant("TRACK_INFO_TRACE")
- CCI_register_constant("TRIEN")
- CCI_register_constant("TRIENB")
- CCI_register_constant("TRISTATE")
- CCI_register_constant("V_FILE")
- CCI_register_constant("v0")
- CCI_register_constant("v1")
- CCI_register_constant("VERBOSE_REPORT")
- CCI_register_constant("VOLTAGE_BLOCK")
- CCI_register_constant("VOLTAGE_REGION")
- CCI_register_constant("vU")
- CCI_register_constant("vX")
- CCI_register_constant("vZ")
- CCI_register_constant("XNOR")
- CCI_register_constant("XOR")
- CCI_register_constant("Y_FILE")

A

Managing VHDL Libraries and Files

Introduction

The concept of libraries is associated more with VHDL than Verilog. This appendix is therefore intended for VHDL users. It presents detailed information on how to manage VHDL resource libraries and files for projects that you create for checking with Leda, in the following major sections:

- [“Setting Libraries” on page 313](#)
- [“Building Libraries” on page 314](#)
- [“Adding Files to VHDL Resource Projects” on page 315](#)
- [“Adding Libraries to VHDL Resource Projects” on page 315](#)
- [“Creating Local VHDL Resource Libraries” on page 316](#)

Setting Libraries

You set libraries when you create a project to organize your HDL source files for checking with Leda (see [“Creating Projects to Check HDL Code” on page 91](#)). Use the Specify Libraries window available from the Project Creation Wizard (**Project > New**), to specify libraries. Specify Libraries is the third window that the Wizard brings you to.

Click the New button at the top of the window to specify the logical name of each working library. Leda creates working libraries in the *project_name*-libs subdirectory. Note that this library is now available both for Verilog and VHDL.

Setting Resource Libraries

Use the Specify Compiler Options window available from the Project Creation Wizard (**Project > New**), to specify resource libraries for VHDL 87 or 93, depending on the version you are using. Specify Compiler Options is the second window that the Wizard brings you to. Just click on the 87 or 93 radio button at the bottom of the window. Leda automatically adds the required resource libraries for your language version.



Caution

You cannot mix VHDL 87 and 93 libraries in the same Leda project.

Use the Specify Libraries window available from the Project Creation Wizard (**Project > Libraries**), to add resource libraries to your project. Specify Libraries is the third window that the Wizard brings you to. Select a logical library name from the pull-down menu at the top of the window. Then use the Add button to navigate to the location of the associated physical library. Click on OK. Leda displays the full path to the new physical library in the window. To remove a physical library from the selected logical library, select the library full path name in the window and click the Remove button.

Building Libraries

When you are done specifying source files and libraries for your project, use the Confirm and Create window in the Project Creation Wizard to build the project and compile your source files. Confirm and Create is the fifth and final window that the Wizard brings you to.

Select the Build with Check check box and click the Finish button at the bottom of the window. This causes Leda to:

- Generate a command file (*project_name.cmd*) and create *project_name*-libs and *project_name*-logs directories in the current working directory for the project.
- Generate a script that creates project libraries in the *project_name*-libs directory.
- Generate a Makefile and use it to compile the source files (in order) into their corresponding libraries in the *project_name*-libs directory.
- Open the project with a name of *project_name.pro*.
- Run the Checker on any policies, or sets of rules, that you have selected.

If the Makefile fails to generate the project correctly, there may be a syntax error in one of the source files. In this case, Leda opens the project with the correct list of files in the Files tab, but in the Modules/Units tab some units are missing. To solve this problem, fix

the syntax errors in your HDL source files and go back to the Specify Source Files window available from the Project Update Wizard (**Project > Edit**). Specify the source files that you fixed and then rebuild the project.

Adding Files to VHDL Resource Projects

Sometimes, it can be useful to add other libraries or units to these resource libraries so that they too are automatically included in every project. For example, most synthesis tools have technology libraries, or tool-specific packages that are stored in the IEEE library. To do this, you must add these libraries to the VHDL 87 or 93 resource projects like you were building a project from scratch. In other words, simply updating the VHDL resource projects does not work. If you want to add a new package to the IEEE library, for example, follow these steps:

1. Place the cursor over the IEEE library icon in the Files tab on the left side of the main window, and hold down the right mouse button.
2. Select the Add Files option and choose the files to add using the Add Files window.



Note

In VHDL, the compilation order for files is important. Leda compiles files in the order that they appear in the Specify Source Files window available from the Project Creation Wizard (**Project > New**) or Update the Project window (**Project > Edit**). Therefore, if a library or file depends on another library or file, it must appear after that library or file in the Specify Source Files window.

All projects that use these VHDL resource libraries now automatically include the new resources.

Adding Libraries to VHDL Resource Projects

You can also add new libraries to one of the default VHDL resource projects. Follow these steps:

1. Place the cursor over the Source Files icon in the Files tab on the left side of the main window, and hold down the right mouse button.
2. Select the Add a Library option and choose the files to add using the Add Working Library window.
3. When the library is created, add and compile the files in the correct compilation order.

4. Choose **Project > Save** to save the project, and **Project > Close** to close the project.

All projects that use these VHDL resource libraries now automatically include the new resources.

Creating Local VHDL Resource Libraries

In some cases, it may not be possible to modify the global resource projects. Perhaps you don't have write permissions or the resources you want to update are not global. In such cases, you can create local resource libraries in a location other than `$LEDA_PATH/resources`. Follow these steps:

1. Set the `LEDA_RESOURCES` environment variable to point to a directory where you have write permissions and want your local VHDL resource libraries to go, as shown in the following example:

```
% setenv LEDA_RESOURCES my_resources_library_directory
```

2. Run the Leda installation setup script and answer Yes when the script asks you if you want to install resource libraries locally, as shown in the following example:

```
% $LEDA_PATH/utilities/setup_custom
```

B

Leda Environment Variables

Introduction

To use Leda, you need to make sure that your environment is set up correctly. You set many Leda environment variables during installation. For information on how to set these variables, see the [Leda Installation Guide](#). Other environment variables are important in the context of specific tasks that you want to perform with Leda. Those are documented in this manual along with the procedures where they apply.

Setting Leda Environment Variables

You can set Leda environment variables in the shell before you invoke the tool, or in a `.synopsys_leda.setup` file that Leda reads at invocation time. Specify values for your environment variables as shown in the following examples:

```
set env(search_path) "your_search_path"
set env(link_library) "your_link_library"
set env(LEDAS_CONFIG) "my_config_dir"
```

Put your `.synopsys_leda.setup` file in `$HOME` or the current working directory (`$cwd`). This file should be used only to set environment variables. Leda uses the last environment variable settings found in the following search path:

- `$HOME/.synopsys_leda.setup`
- `$cwd/.synopsys_leda.setup`

Using Leda Environment Variables

All of the Leda environment variables and their uses are also listed in [Table 36](#). Note that you can check your environment at any time while using the Leda GUI by clicking on the Info Report tab on the right side of the main window. For more information on using the Info Report, see [“Checking Your Environment” on page 171](#).

Table 36: Leda Environment Variables

Environment Variable	Use
HTML_NAVIGATOR	Set this variable to the location of the HTML browser that you want Leda to use for viewing HTML-based help files and reports.
LEDA_CONFIG	Set this variable to the location of a global configuration not in \$LEDA_PATH, in cases where multiple users are expected to use the same prepackaged rule configuration when checking their designs. For more information, see “Configuring the Rule Wizard” on page 73 .
LEDA_CLOCK_FILE	Set this variable to the dumped modified clock file. The CDC rules take these information as inputs. For more information, see “Clock Grouping Feature” on page 66 .
LEDA_HTML_DOC_PATH	If you write your own custom rules, set this variable to point to the directory where the primary HTML-based help file is located.
LEDA_HTML_USR_PATH	If you write your own custom rules, set this variable to point to the directory where the secondary HTML-based help file is located. Use the secondary help file for additional information or application notes for each rule.
LEDA_LANGUAGE	If you want to see messages in the Error Viewer for the VER_STARC_2001 and VHD_STARC_2001 policies displayed in Japanese instead of English (the default), set this variable to JAPANESE before invoking Leda. For more information, see “Displaying Error Messages for STARC Policies” on page 115 .
LEDA_MAX_CLOCKS	Set this variable to define the maximum clock limit when using the clock file feature. The default value is 500. For more information, see “Clock Grouping Feature” on page 66 .
LEDA_PATH	Set this variable to the directory where you installed Leda before you try to run the software. You also set this variable to the target directory before you run the custom installation script (setup_custom).
LEDA_READER	Set this variable to the location of the PDF file reader that you want Leda to use for viewing the PDF-based Leda documentation. (Acrobat Reader recommended.)

Table 36: Leda Environment Variables (Continued)

Environment Variable	Use
LEDA_RESOURCES	For VHDL only. Set this variable to the directory where you want to optionally install a copy of the IEEE VHDL resource libraries. This local installation is in addition to the required global installation. Set before running the setup_custom installation script (see “Creating Local VHDL Resource Libraries” on page 316).
LEDA_SELECT_FILE	In older versions of Leda, this variable was used to point to the location of a .leda_select file that you could use to deactivate rules. For more information, see “Translating .leda_select Files” on page 106.
link_library	Set this variable to the location of any technology-dependent .db files that you want Leda to search when trying to resolve architecture/model instantiations for checking chip-level rules. For more information, see “Using .db Files for Checks” on page 39. You can specify multiple link_libraries by separating the names with spaces and enclosing the list of entries in quotation marks. For example: <pre>% setenv link_library "gtech.db class.db"</pre>
search_path	Set this variable to the location of technology-dependent .db libraries if you want Leda to search these libraries during the analysis and elaboration of designs for checking chip-level rules. See “Using .db Files for Checks” on page 39. You can specify multiple search_paths by separating the names with spaces and enclosing the list of entries in quotation marks. For example: <pre>% setenv search_path "/u/me/lib1 /u/me/lib2"</pre> Not that the search_path variable can only be used to find link libraries, not design files.
LM_LICENSE_FILE	Standard FLEXlm license file variable. Set to the full path to your license file (license.dat) or license server (<i>port@host</i>).
SNPSLMD_LICENSE_FILE	Alternative Synopsys license file variable. Set to the full path to your license file (license.dat) or license server (<i>port@host</i>).

C

Leda Prebuilt Configurations

Overview

Leda supports eight different prebuilt rule configurations. Each section in this appendix lists the rules contained in these configurations and explains how to load them:

- [“RTL Prebuilt Configuration” on page 322](#)
- [“Gate-level Prebuilt Configuration” on page 325](#)
- [“Leda-classic Prebuilt Configuration” on page 327](#)
- [“CDC Prebuilt Configuration” on page 387](#)
- [“SDC-postlayout Prebuilt Configuration” on page 388](#)
- [“SDC-prelayout Prebuilt Configuration” on page 390](#)
- [“SDC-RTL Prebuilt Configuration” on page 393](#)
- [“SDC-top-versus-block Prebuilt Configuration” on page 396](#)
- [“SDC-equivalency Prebuilt Configuration” on page 397](#)



Note

Leda-optimized is a subset of the Leda-classic prebuilt configuration. This configuration is “optimized” to remove similar rules from different policies.

RTL Prebuilt Configuration

The following rules are from the RTL prebuilt configuration. This configuration contains about 70 rules drawn from the DC, DFT, Formality, RMM, and Leda general coding guidelines policies. This configuration is the default. To load this rule configuration, from the Rule Wizard, choose **Config > Load configuration**, and use the pull-down menu to select RTL.

Table 37: RTL Prebuilt Configuration

Rule Label	Policy	Message
DCHDL_115	DC	Illegal mixing of named and unnamed port association.
DCHDL_178	DC	Only simple variables are checked in the sensitivity list.
DCVER_192	DC	Initial statement not supported.
DCVER_274	DC	Verilog system task is not supported.
DCVHDL_165	DC	'while' statement not supported.
DFT_021	DFT	Latch inferred for <%item>.
DFT_022	DFT	Incomplete case statement.
FM_2_10	FORMALITY	Using X, Z values or ? in case items is not recommended (such items may be ignored by synthesis tools).
FM_2_12	FORMALITY	Incomplete case_statement using full_case directive is not recommended (not supported by some emulation tools).
FM_2_13	FORMALITY	When case items are duplicated (parallel), do not use parallel_case directive.
FM_2_18	FORMALITY	Case choice after the default may be ignored by some simulation tools.
FM_2_4	FORMALITY	Assignment to X is not recommended (handled differently by simulation and synthesis tools).
FM_2_7	FORMALITY	Use named association in port map.
FM_2_9	FORMALITY	Using X, Z values or ? for comparison is not recommended (differently handled by simulation/synthesis tools).
B_1001	LEDA	Reading from outputport <%item>.
B_1002	LEDA	Port declaration <%item> is unused or partially unused.
B_1011	LEDA	Module instantiation is not fully bound. Port <%format> is not completely connected.

Table 37: RTL Prebuilt Configuration (Continued)

Rule Label	Policy	Message
B_1204	LEDAs	Multi-bit expression used as a clock.
B_2001	LEDAs	Shift by a non constant value is not allowed.
B_2011	LEDAs	Variable is not always initialized in process body before being read.
B_3010	LEDAs	Loop index must be declared as integer.
B_3203	LEDAs	The expression in for loop must not be constant.
B_3208	LEDAs	Unequal length LHS and RHA in assignment.
B_3209	LEDAs	Unequal length port and connection in module instantiation.
B_3408	LEDAs	Case condition should not be constant.
B_3409	LEDAs	While condition expression is constant.
B_3410	LEDAs	X in case expression.
B_3416	LEDAs	Use blocking assignments in combinatorial block.
B_3417	LEDAs	Use non-blocking assignments in sequential block.
B_3419	LEDAs	Missing signal <%item> in sensitivity list.
B_3602	LEDAs	Moore style description of state machines is recommended.
B_3604	LEDAs	Assign a default state to the state machines.
B_3605_A	LEDAs	Use parameter declarations to define the state vector of a state machine.
B_3605_B	LEDAs	Use an enumerated type to define the state vector of a state machine.
B_3607	LEDAs	The number of states in a state machine should be a power of 2.
C_1000	LEDAs	Asynchronous feedback loop detected.
C_1001	LEDAs	Flip-flop with fixed value data input is detected.
C_1005	LEDAs	Top-level outputs are not registered.
C_1007	LEDAs	Pulse generator detected.
C_1009	LEDAs	Multiple non-tristate drivers to signal <%item> detected.
C_1201	LEDAs	Clocks must not be used as data.

Table 37: RTL Prebuilt Configuration (Continued)

Rule Label	Policy	Message
C_1203	LEDA	Internally generated clock detected (chip level).
C_1204	LEDA	No gated clock except in clock generator CKGEN.
C_1406	LEDA	Register with no reset/set/load is detected.
G_546_1	RMM_RTL_ CODING_ GUIDE- LINES	Avoid internally generated reset/load <%item>.
G_551_1_B	RMM_RTL_ CODING_ GUIDE- LINES	The always keyword must be followed by an event list @(...) in a sequential block.
R_521_10	RMM_RTL_ CODING_ GUIDE- LINES	Always use descending range for multi-bit signals and ports.
R_529_1	RMM_RTL_ CODING_ GUIDE- LINES	VHDL or Verilog reserved words cannot be used as identifiers.

Gate-level Prebuilt Configuration

The following rules are from the Gate-level prebuilt configuration. This configuration contains 90 chip-level and netlist/design rules selected from the Design and Leda general coding guidelines policies. To load this rule configuration, from the Rule Wizard, choose **Config > Load configuration**, and use the pull-down menu to select Gate-level.

Table 38: Gate-level Prebuilt Configuration

Rule Label	Policy	Message
NTL_CLK04	DESIGN	Do not use internally generated clock.
NTL_CLK05	DESIGN	All asynchronous inputs to a clock system must be clocked twice.
NTL_CLK07	DESIGN	Avoid gated clocks unless absolutely necessary.
NTL_CLK13	DESIGN	Buffer on clock path detected.
NTL_CLK14	DESIGN	Inverter on clock path detected.
NTL_CLK17	DESIGN	Reconvergent path on clock tree detected.
NTL_CLK21	DESIGN	Pulse generator created by self flip-flop.
NTL_CLK22	DESIGN	Clock chopper/extender detection.
NTL_CON01	DESIGN	Unconnected top level input port.
NTL_CON02	DESIGN	Unconnected top level output port.
NTL_CON03	DESIGN	Unconnected top level inout port.
NTL_CON06	DESIGN	Input pin tied to supply.
NTL_CON10	DESIGN	Output tied to supply.
NTL_PAD09	DESIGN	Forbidden pad connection.
NTL_PAD11	DESIGN	Isolate I/O pad from the core logic.
NTL_PAR19	DESIGN	Clock generation logic should be put in a particular module.
NTL_RST04	DESIGN	A reset signal is not allowed to be used as an input to control path logic.
NTL_RST06	DESIGN	Avoid internally generated resets.
NTL_RST07	DESIGN	Don't use one reset signal for both asynchronous reset and synchronous reset.

Table 38: Gate-level Prebuilt Configuration (Continued)

Rule Label	Policy	Message
NTL_RST12	DESIGN	Buffer on reset path detected.
NTL_RST13	DESIGN	Inverter on reset path detected.
NTL_RST16	DESIGN	Reconvergent path on reset tree detected.
NTL_STR05	DESIGN	A signal that passes through several hierarchical levels must have the same name throughout.
NTL_STR07	DESIGN	Avoid glue logic at top level.
NTL_STR18	DESIGN	Avoid clock as set or reset circuitry.
NTL_STR23	DESIGN	Max. number of fanout between modules.
NTL_STR24	DESIGN	Number of logical levels between 2 flip-flops exceeds maximum limit.
NTL_STR37	DESIGN	Avoid combinatorial logic on the control signal of a tristate driver.
NTL_STR47	DESIGN	Do not use latch.
NTL_STR61	DESIGN	Do not use clock or enable signals as data inputs.
C_1000	LEDA	Asynchronous feedback loop detected
C_1001	LEDA	Flip-flop with fixed value data input is detected.
C_1003	LEDA	Latch detected in design (inferred or instantiated).
C_1004	LEDA	Glue logic at top-level is detected.
C_1005	LEDA	Top-level outputs are not registered.
C_1006	LEDA	Top-level inputs are not registered.
C_1007	LEDA	Pulse generator detected.
C_1009	LEDA	Multiple non-tristate drivers to signal <%item> detected.

Leda-classic Prebuilt Configuration

The following rules are from the Leda-classic prebuilt configuration. This configuration is close to the default configuration used in older versions of the tool. To load this rule configuration, from the Rule Wizard, choose **Config > Load configuration**, and use the pull-down menu to select Leda-classic.



Note

Leda-optimized is a subset of the Leda-classic prebuilt configuration. This configuration is “optimized” to remove similar rules from different policies.

Table 39: Leda-classic Prebuilt Configuration

Rule Label	Policy	Message
DCHDL_109	DC	This use of clock edge specification not supported.
DCHDL_115	DC	Illegal mixing of named and unnamed port association.
DCHDL_170	DC	Comparisons to a unknown, three-state are treated as always being false. This may cause simulation to cause to disagree with synthesis.
DCHDL_175	DC	Clock variable is being used as data
DCHDL_177	DC	Local variable is being read before its value is assigned. This may cause simulation not to match synthesis.
DCHDL_178	DC	Only simple variables are checked in the sensitivity list.
DCHDL_224	DC	Wait statements in process use different clocks or clock edges.
DCHDL_230	DC	Package name <%item> is an internal package. Please use a different name for your package.
DCHDL_270	DC	An unsupported expression is assigned to constant.
DCHDL_326	DC	Enumeration type defined in a generate statement is not supported.
DCHDL_389	DC	Name too long for compiled code.
DCHDL_6	DC	Loop body will iterate zero times.
DCHDL_96	DC	Infinite recursion detected.
DCVER_129	DC	Intra-assignment delays for blocking statements are ignored.
DCVER_131	DC	This design contains event in verilog blocking assignment.
DCVER_132	DC	This design contains event in verilog non-blocking assignment.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
DCVER_135	DC	Intra-assignment repeat-event controls for non-blocking assignments are ignored.
DCVER_143	DC	RTL assignments are allowed only when no blocking delays are used.
DCVER_154	DC	Gate instance with too few ports. Port <%format> is not completely connected.
DCVER_173	DC	Delays for continuous assignment are ignored.
DCVER_176	DC	Delay statements are ignored for synthesis.
DCVER_177	DC	REAL declarations are not supported by synthesis.
DCVER_178	DC	REAL TIME declarations are not supported by synthesis.
DCVER_179	DC	TRIAND declarations are not supported by synthesis.
DCVER_180	DC	TRIOR declarations are not supported by synthesis.
DCVER_181	DC	TRIO declarations are not supported by synthesis.
DCVER_182	DC	TRII declarations are not supported by synthesis.
DCVER_183	DC	TRIREG declarations are not supported by synthesis.
DCVER_184	DC	PULLDOWN declarations are not supported by synthesis.
DCVER_185	DC	PULLUP declarations are not supported by synthesis.
DCVER_187	DC	FORK and JOIN constructs are not supported by synthesis.
DCVER_188	DC	WAIT statements are not supported by synthesis.
DCVER_189	DC	CASE EQUALITY (===) is not supported by synthesis.
DCVER_190	DC	CASE INEQUALITY (===) is not supported by synthesis.
DCVER_191	DC	TIME declarations are not supported.
DCVER_192	DC	Initial statement not supported.
DCVER_193	DC	Event triggers not supported.
DCVER_219	DC	Repeat constructs are not supported in synthesis.
DCVER_256	DC	Illegal part selection.
DCVER_265	DC	RCMOS switches are not supported.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
DCVER_266	DC	RNMOS switches are not supported.
DCVER_267	DC	RPMOS switches are not supported.
DCVER_268	DC	RTRAN switches are not supported.
DCVER_269	DC	RTRANIF0 switches are not supported.
DCVER_270	DC	RTRANIF1 switches are not supported.
DCVER_271	DC	TRAN switches are not supported.
DCVER_272	DC	TRANIF0 switches are not supported.
DCVER_273	DC	TRANIF1 switches are not supported.
DCVER_274	DC	Verilog system task is not supported.
DCVER_275	DC	User-defined primitives (UDPs) are not supported.
DCVER_276	DC	Specify blocks are not supported.
DCVER_277	DC	Charge strengths are ignored.
DCVER_286	DC	EVENT declarations are not supported.
DCVER_295	DC	CMOS switches are not supported.
DCVER_296	DC	NMOS switches are not supported.
DCVER_297	DC	PMOS switches are not supported.
DCVER_305	DC	Drive strength specification for gate instances are ignored.
DCVER_306	DC	Drive strength specification for tristate gate instantiation is ignored.
DCVER_309	DC	Drive strength specification for continuous assignment is ignored.
DCVER_310	DC	Keyword 'scalared' is ignored.
DCVER_311	DC	Parameter range specification is only meaningful to synthesis. Synthesis and simulation may have different results.
DCVER_4	DC	Incompatible port connection in module instantiation.
DCVER_91	DC	Module contains a supply 1 variable. Replace with wire driven by continuous assignment to 1.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
DCVER_917	DC	The 'inout' port <%item> is incompatibly declared as real.
DCVER_919	DC	The 'input' port <%item> is incompatibly declared as real.
DCVER_966	DC	Procedural-continuous assignments are not supported by synthesis.
DCVER_967	DC	The 'force' construct is not supported by synthesis.
DCVER_968	DC	The 'release' construct is not supported by synthesis.
DCVER_969	DC	The 'deassign' construct is not supported by synthesis.
DCVER_970	DC	The delay specification for gate instantiation is ignored.
DCVER_971	DC	The delay specification for tristate gate instantiation is ignored.
DCVER_972	DC	The delay specification for MOS switch instantiation is ignored.
DCVER_973	DC	The delay specification for cmos switch instantiation is ignored.
DCVER_974	DC	The delay specification for bidirectional switch instantiation is ignored.
DCVER_976	DC	The delay specification for net declaration is ignored.
DCVER_977	DC	The strength specification for a net declaration is ignored by synthesis.
DCVHDL_104	DC	'SIGNAL' declaration for subprogram input port ignored.
DCVHDL_111	DC	GUARDED is not supported. It is ignored.
DCVHDL_160	DC	'OTHERS' and 'ALL' not supported for attribute specification.
DCVHDL_165	DC	'while' statement not supported.
DCVHDL_179	DC	Iteration scheme required.
DCVHDL_197	DC	Assumed to be of type 'integer'
DCVHDL_2001	DC	Statements in an entity declaration are not supported for synthesis. They are ignored.
DCVHDL_2021	DC	'BUS' and 'REGISTER' signal kinds are not supported for synthesis.
DCVHDL_2022	DC	Initial valued for signals are not supported for synthesis. They are ignored.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
DCVHDL_ 2023	DC	Type of the generic is assumed to be 'Integer' in synthesis.
DCVHDL_ 2024	DC	Only generics of type INTEGER are supported for synthesis.
DCVHDL_ 2040	DC	Attribute not supported for synthesis.
DCVHDL_ 2041	DC	Alias declarations are not supported for synthesis. They are ignored.
DCVHDL_ 2042	DC	File declarations are not supported for synthesis. They are ignored.
DCVHDL_ 2043	DC	Disconnection specifications are not supported for synthesis. They are ignored.
DCVHDL_ 2045	DC	Guard conditions for blocks are not supported.
DCVHDL_ 2046	DC	Declaration and use of generics and ports in a block header is not supported.
DCVHDL_ 2050	DC	Timeout clause not supported for synthesis in wait statement.
DCVHDL_ 2090	DC	Declarations in a configuration declaration statement are not supported for synthesis. They are ignored.
DCVHDL_ 2091	DC	Configuration specifications are not supported for synthesis.
DCVHDL_ 2092	DC	Only simple configurations (specification of architecture for a top-level entity) are supported for synthesis. Nested block specifications and component configurations are ignored.
DCVHDL_ 2093	DC	Access types are not supported for synthesis.
DCVHDL_ 2094	DC	File types are not supported for synthesis. They are ignored.
DCVHDL_ 2095	DC	Physical types are not supported for synthesis. They are ignored.
DCVHDL_ 2096	DC	Incomplete type declarations are not supported for synthesis. They are ignored.
DCVHDL_ 2097	DC	Signal assignment delays are not supported for synthesis. They are ignored.
DCVHDL_ 2098	DC	'Transport' construct is not supported for synthesis. It is ignored.
DCVHDL_ 2099	DC	Assert and report statements are not supported for synthesis. They are ignored.
DCVHDL_ 2100	DC	Allocators are not supported for synthesis.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
DCVHDL_ 2108	DC	Wait statements are not supported in subprograms.
DCVHDL_ 2109	DC	Event and Stable attributes are not supported in subprograms.
DCVHDL_ 2111	DC	Aggregate assignment by name is not supported for the field names of a record.
DCVHDL_ 2131	DC	Configurations are not supported for direct instantiation during synthesis.
DCVHDL_ 2140	DC	Multi-dimensional arrays are not supported for synthesis.
DCVHDL_ 2150	DC	This form of wait statement is not supported for synthesis.
DCVHDL_ 2151	DC	Attribute is not supported for synthesis.
DCVHDL_ 2152	DC	This literal is not supported for synthesis.
DCVHDL_ 2155	DC	Deferred constants are not supported for synthesis.
DCVHDL_ 2159	DC	Empty string constants are not supported for synthesis.
DCVHDL_ 2163	DC	The rising_edge or falling_edge function is supported only when used in conformance with the style described in the VHDL Reference Manual.
DCVHDL_ 2207	DC	You have declared a component inside a for generate loop.
DCVHDL_ 2251	DC	Enabling expression not permitted outside wait statements.
DCVHDL_ 2254	DC	Time is an unsupported type.
DCVHDL_ 2255	DC	Generics of type string are not supported.
DCVHDL_ 2262	DC	Enumeration values may not be used as for or for-generate loop bounds.
DCVHDL_ 2264	DC	Incorrect way to use attribute.
DCVHDL_ 2270	DC	Aliases to existing aliases are not supported for synthesis.
DCVHDL_ 228	DC	Initial values are not supported for variables.
DCVHDL_ 2284	DC	Declarative regions of generate statements is not supported.
DCVHDL_ 279	DC	STD.TEXTIO package is not supported for synthesis.
DC_31	DC	Static data types are not supported in \$root
DC_39	DC	Array literals are not supported.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
DC_42	DC	Casting is not supported.
DC_53	DC	Import/export of tasks and functions is not supported.
DC_54	DC	Process statement is not supported.
DC_55	DC	Nested module/interface declaration is not supported.
NTL_CLK01	DESIGN	Use only one clock domain
NTL_CLK03	DESIGN	Use only one edge of the clock.
NTL_CLK04	DESIGN	Do not use internally generated clock.
NTL_CLK05	DESIGN	All asynchronous inputs to a clock system must be clocked twice.
NTL_CLK07	DESIGN	Avoid gated clocks unless absolutely necessary.
NTL_CLK08	DESIGN	If gated clocks are necessary, isolate them and make them global.
NTL_CLK09	DESIGN	All clock signals should be generated in a module driven by a single external clock.
NTL_CLK10	DESIGN	Clock signal gated with an OR gate.
NTL_CLK11	DESIGN	Clock signal gated with an AND gate.
NTL_CLK12	DESIGN	Clock signal gated with another combinatorial cell.
NTL_CLK13	DESIGN	Buffer on clock path detected.
NTL_CLK14	DESIGN	Inverter on clock path detected.
NTL_CLK15	DESIGN	Clock pin not connected to clock net.
NTL_CLK17	DESIGN	Reconvergent path on clock tree detected.
NTL_CLK18	DESIGN	Try to concentrate the clock generation circuitry at the top-level of the design.
NTL_CLK19	DESIGN	Do not add XOR or XNOR on clock path.
NTL_CLK20	DESIGN	Use only OR gate for joined clock.
NTL_CLK21	DESIGN	Pulse generator created by self flip-flop.
NTL_CLK22	DESIGN	Clock chopper/extender detection.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
NTL_CLK23	DESIGN	Multiple asynchronous clock domain signals converging on <gate name>
NTL_CON01	DESIGN	Unconnected top level input port.
NTL_CON02	DESIGN	Unconnected top level output port.
NTL_CON03	DESIGN	Unconnected top level inout port.
NTL_CON04	DESIGN	All inputs pin tied together.
NTL_CON06	DESIGN	Input pin tied to supply.
NTL_CON10	DESIGN	Output tied to supply.
NTL_CON15	DESIGN	Power rails belonging to different supply types should not short.
NTL_CON16	DESIGN	Nets or cell pins should not be tied to logic 0/logic 1.
NTL_CON17	DESIGN	Do not connect tie off cell to logic 0/logic 1.
NTL_DFT02	DESIGN	Separate DFT functionality from regular functionality.
NTL_DFT07	DESIGN	Internally generated output enable signal must be observable/controllable.
NTL_DFT08	DESIGN	No contention may take place during scan-test mode.
NTL_DFT09	DESIGN	Scan-input and scan output must be a primary input/output.
NTL_DFT10	DESIGN	Scan chain too long.
NTL_DFT11	DESIGN	Flip-flops in a scan chain must have a common scan clock.
NTL_DFT12	DESIGN	Separate scan chain for different clock domain.
NTL_DFT13	DESIGN	Use a single clock edge for a given scan-chain.
NTL_DFT14	DESIGN	Use a single clock for a given scan-chain.
NTL_DFT15	DESIGN	All scan-in input port must be controllable from the top.
NTL_DFT16	DESIGN	All scan-out output port must be observable from the top.
NTL_DFT17	DESIGN	Scan-in used in combinatorial part.
NTL_DFT22	DESIGN	Use one synchronous clock (positive or negative edge) during test.
NTL_DFT23	DESIGN	Clock signal must be controllable.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
NTL_DFT24	DESIGN	Internally generated clock must be observable/controllable.
NTL_DFT25	DESIGN	Make sure that all sequential scan-cells are driven with a test clock while the scan mode is active.
NTL_DFT26	DESIGN	Scan clock control flip-flop data.
NTL_DFT27	DESIGN	Scan clock control I/O cell.
NTL_DFT28	DESIGN	Scan clock should be called scan_clk...
NTL_DFT29	DESIGN	Asynchronous set/reset inputs of flip-flops must be inactive during scantest.
NTL_DFT32	DESIGN	Connect all non-observable nodes of the design to a Xor tree.
NTL_DFT34	DESIGN	Use data-look-up latches for clock domain crossing.
NTL_DFT36	DESIGN	Do not use scan-type flip flops for functional mode.
NTL_DFT37	DESIGN	Insert scan flip-flop around Black box.
NTL_DFT41	DESIGN	Flip-flop with tied input is detected.
NTL_DFT50	DESIGN	All sequential cells must be connected to a test clock in test mode.
NTL_DFT52	DESIGN	All set/reset pins must be controllable during test mode.
NTL_DFT53	DESIGN	Scan-enable must be controllable from the top.
NTL_DFT54	DESIGN	Insert test enable for scan test.
NTL_DFT55	DESIGN	Test-enable signal must be generated from the scan-mode or directly from the primary inputs.
NTL_DFT56	DESIGN	Scan-enable used in combinatorial part.
NTL_LAN01	DESIGN	Tri assignment target must be a port.
NTL_LAN15	DESIGN	Unused signals.
NTL_LAN21	DESIGN	Netlist and libraries shall not have upper-lower case clash.
NTL_LAM02	DESIGN	Clock ports must be assigned to internal clock signals that start with clk.
NTL_LAM03	DESIGN	A reset port must be assigned to an internal reset signal that starts with rst.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
NTL_LAM07	DESIGN	A registered output port should end with _r.
NTL_PAD05	DESIGN	Controllable pull-up.
NTL_PAD06	DESIGN	Controllable pull-down.
NTL_PAD07	DESIGN	Push pull always disabled.
NTL_PAD08	DESIGN	Input port always disabled.
NTL_PAD09	DESIGN	Forbidden pad connection.
NTL_PAD10	DESIGN	Output port always disabled.
NTL_PAD11	DESIGN	Isolate I/O pad from the core logic.
NTL_PAD13	DESIGN	Primary inputs must be connected to exactly 1 PAD cell.
NTL_PAR13	DESIGN	Separate the design according to clock domains.
NTL_PAR18	DESIGN	Clock and Reset generators should be located at the top of the design in a dedicated module.
NTL_PAR19	DESIGN	Clock generation logic should be put in a particular logic.
NTL_RST01	DESIGN	Use only one reset domain.
NTL_RST02	DESIGN	A system reset must be defined.
NTL_RST04	DESIGN	A reset signal is not allowed to be used as an input to control path logic.
NTL_RST05	DESIGN	Don't use asynchronous set/reset signal except for initial reset.
NTL_RST06	DESIGN	Avoid internally generated resets.
NTL_RST07	DESIGN	Don't use one reset signal for both asynchronous reset and synchronous reset.
NTL_RST08	DESIGN	Locally gated asynchronous resets should be avoided.
NTL_RST09	DESIGN	Reset signal gated with an OR gate.
NTL_RST10	DESIGN	Reset signal gated with an AND gate.
NTL_RST11	DESIGN	Reset signal gated with another combinatorial cell.
NTL_RST12	DESIGN	Buffer on reset path detected.
NTL_RST13	DESIGN	Inverter on reset path detected.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
NTL_RST14	DESIGN	Reset pin not connected to reset net.
NTL_RST16	DESIGN	Reconvergent path on reset tree detected.
NTL_RST17	DESIGN	Reset gating must take care of the flip-flop triggering edge. Flip-flop must be of opposite edge.
NTL_RST18	DESIGN	Reset signal must not interact with the other latch pins.
NTL_SET01	DESIGN	Use only one set domain.
NTL_STR02	DESIGN	Avoid asynchronous design.
NTL_STR04	DESIGN	Enable signals of bidirectional PADs should be registered before being connected to an output.
NTL_STR05	DESIGN	A signal that passes through several hierarchical levels must have the same name throughout.
NTL_STR06	DESIGN	Top level output should be registered.
NTL_STR07	DESIGN	Avoid glue logis at top level.
NTL_STR08	DESIGN	Use gate instantiation only at a few instances.
NTL_STR11	DESIGN	VDD and GND must not be fed directly into logic.
NTL_STR14	DESIGN	Check that circuits labeled <code>_meta</code> are really proper metastable circuits.
NTL_STR15	DESIGN	Give unique name to synchronizers so that they can be identified.
NTL_STR16	DESIGN	Do not use bidirectional ports in sub-modules of your design.
NTL_STR18	DESIGN	Avoid <code>clk</code> as set or reset circuitry.
NTL_STR19	DESIGN	Detected multiply driven signal.
NTL_STR20	DESIGN	Each output enable signal should be assigned to no more than
NTL_STR21	DESIGN	The level of the design hierarchy should not exceed %d: %s.
NTL_STR22	DESIGN	Inhibit: Use of black box.
NTL_STR23	DESIGN	Max number of fanout between modules.
NTL_STR24	DESIGN	Number of logical levels between 2 flip-flops exceeds maximum limit.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
NTL_STR26	DESIGN	No INOUTs at any top level block, although acceptable they consume a lot of resources in the box.
NTL_STR27	DESIGN	Parallel inverters.
NTL_STR28	DESIGN	Delay line.
NTL_STR29	DESIGN	Pulse generator.
NTL_STR30	DESIGN	Shift registers.
NTL_STR31	DESIGN	Netlist not uniquified.
NTL_STR34	DESIGN	No internal three-state buffers are allowed.
NTL_STR37	DESIGN	Avoid combinatorial logic on the control signal of a tri-state driver.
NTL_STR43	DESIGN	Use template for inferred tri-state buffer.
NTL_STR45	DESIGN	Single tri-state detected.
NTL_STR47	DESIGN	Do not use latch.
NTL_STR48	DESIGN	Latches shall be instantiated using the VLSI generic latch components.
NTL_STR50	DESIGN	Inhibit: Latch to Latch path detected.
NTL_STR51	DESIGN	Inhibit: Latch with set and reset.
NTL_STR53	DESIGN	Anti-skew latch enable not controlled by main clock.
NTL_STR54	DESIGN	The enable signal of tri-state or bidirectional ports must be available at the core boundary.
NTL_STR55	DESIGN	Do not use bidirectional ports for scan enable.
NTL_STR56	DESIGN	Do not use sequential registers with both asynchronous set and asynchronous reset.
NTL_STR57	DESIGN	Inhibit: multiple asynchronous reset (set) signal.
NTL_STR58	DESIGN	Register controlled b multiple clock.
NTL_STR59	DESIGN	Inout signal connect to register clock.
NTL_STR61	DESIGN	Do not use clock or enable signals as data inputs.
NTL_STR62	DESIGN	Netlist shall not have parallel drive situations.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
NTL_STR63	DESIGN	Tristate and non tristate drivers are driving the same net.
NTL_STR65	DESIGN	Number of buffers/inverters should not exceed a user specified percentage of total cell count.
NTL_STR66	DESIGN	Do not instantiate big buffers.
NTL_STR67	DESIGN	Max. number of flip-flop that belong to one clock domain.
NTL_STR68	DESIGN	Don't use that cell.
NTL_STR69	DESIGN	Do not use feedthrough.
NTL_STR70	DESIGN	Set and reset signal must not come from a common source.
NTL_STR72	DESIGN	A non-tristate net can have only one non-tristate driver.
NTL_STR73	DESIGN	A tristate net can have exactly 1 bus keeper cell.
NTL_STR74	DESIGN	A non tristate net can have zero bus keeper.
NTL_STR75	DESIGN	Different Vt cells used.
NTL_STR83	DESIGN	Use only parallel connections that are supported by PrimeTime.
NTL_STR84	DESIGN	Latch enabled by a clock feeds latches enabled by the same clock.
DFT_002	DFT	Internally generated clock detected.
DFT_003	DFT	Avoid using both positive-edge and negative-edge triggered flip-flops in your design
DFT_006	DFT	<% value> clocks in block.
DFT_008	DFT	Tri-state is detected.
DFT_009	DFT	Register all outputs from the block for improved coverage: %s
DFT_017	DFT	Synchronous reset/set/load <%item> detected.
DFT_019	DFT	Asynchronous reset/set/load <%item> detected.
DFT_021	DFT	Latch inferred for <%item>
DFT_022	DFT	Incomplete case statement.
TEST_953	DFT	Flip-flops with clocks tied to a signal that is not driven by Test Clock. Flip-flops' clock signal is not reached by any signal.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
TEST_954	DFT	Latches with clocks tied to a signal that is not driven by Test Clock. Latch clock signal is not reached by any signal.
TEST_960	DFT	Avoid asynchronous feedback loops.
TEST_963	DFT	Flip-flops have clock with no off-state controllability. Test clock reaches flip-flops but does not control them at beginning of cycle.
TEST_964	DFT	Latches have clock with no off-state controllability. Test clock reaches latches but does not control them at beginning of cycle.
TEST_965	DFT	Latches not holding data in off-state. Test Clock reaches latch <%item> but does not hold data at beginning of cycle.
TEST_966	DFT	Flip-flops have no asynch controllability. No Test Asynch reaches flip-flops' asynch control pin.
TEST_967	DFT	Latches have no asynch controllability. No Test Asynch reaches latches asynch control pin.
TEST_968	DFT	Flip-flops have asynchs that cannot be disabled. Test Asynch reaches flip-flops but cannot disable their asynch controls.
TEST_969	DFT	Latches have asynchs that cannot be disabled. Test Asynch reaches latches but cannot disable their asynch controls.
TEST_970	DFT	Clock affects data inputs of flip-flops.
TEST_971	DFT	Clock affects data inputs of latches.
TEST_972	DFT	Clock affects both clock and data inputs of flip-flops.
TEST_973	DFT	Clock affects both clock and data inputs of latches.
TEST_974	DFT	Latch enabled by a clock feeds latches enabled by the same clock.
TEST_975	DFT	Latch enabled by a clock affects data input of flip-flops clocked by the trailing edge of the same clock.
TEST_976	DFT	Latches capture only when more than one clock is on.
TEST_977	DFT	Flip-flops capture only when more than one clock is on.
TEST_978	DFT	Latch data gates clocks of flip-flops. Combination of latch data and clock signal to clock a flip-flop is not allowed.
TEST_979	DFT	Latch data gates clocks enabling latches. Combination of latch data and clock signal to clock a latch is not allowed.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
TEST_980	DFT	Flip-flop data gates clocks to flip-flops. Combination of flip-flop data and clock signal to clock a flip-flop is not allowed.
TEST_981	DFT	Flip-flop data gates clocks enabling latches. Combination of flip-flop data and clock signal to clock a latch is not allowed.
TEST_994	DFT	Clock affects multiple clock or async ports of register.
FM_106	FORMALITY	Do not use power operator.
FM_108	FORMALITY	Do not use recursive task or function.
FM_111	FORMALITY	Do not use v2k enhanced file IO.
VLOG_038	FORMALITY	Do not use variable initial value.
FM_1_1	FORMALITY	Avoid asynchronous feedback loops.
FM_2_10	FORMALITY	Using X,Z values or ? in case items is not recommend (such items may be ignored by synthesis tools).
FM_2_11	FORMALITY	Using signals in casex/z items is not recommended (may be treated as don't care by simulation tools).
FM_2_12	FORMALITY	Incomplete case_statement using full_case directive is not recommended (not supported by some simulation tools).
FM_2_13	FORMALITY	When case items are duplicated (parallel) do not use parallel_case directive.
FM_2_15	FORMALITY	Using blocking assignments in sequential always block may generate incorrect logic.
FM_2_16	FORMALITY	Using non-blocking assignments in combinational always block may generate incorrect logic.
FM_2_17	FORMALITY	Avoid operand size mismatch assignments.
FM_2_18	FORMALITY	Case choice after the default may be ignored by some simulation tools.
FM_2_19	FORMALITY	Using net type other than wire (wand, wor, ...) is not recommended (can generate mismatch during simulation).
FM_2_1A	FORMALITY	Redundant signal <%item> in the sensitivity list.
FM_2_1B	FORMALITY	Missing signal <%item> in the sensitivity list.
FM_2_2	FORMALITY	Delays are ignored by synthesis tools.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
FM_2_20	FORMALITY	Do not use event_control in assignments (not handled by all tools).
FM_2_21	FORMALITY	Do not use duplicated port definitions (some tools rename duplicated ports automatically).
FM_2_22	FORMALITY	Possible range overflow.
FM_2_23	FORMALITY	Non driven output ports or signals <%context> detected.
FM_2_24	FORMALITY	Bit/part select signals detected in sensitivity list: may be ignored by some synthesis and simulation tools.
FM_2_25	FORMALITY	Operator === is treated as ==.
FM_2_26	FORMALITY	Operator !== is treated as !=.
FM_2_27	FORMALITY	Keyword TRANSPORT is ignored in signal assignment.
FM_2_3	FORMALITY	Variables must be initialized before being used. (to prevent latch inference).
FM_2_32	FORMALITY	Do not use latch description in subprogram.
FM_2_4	FORMALITY	Assignment to X is not recommended (handled differently by synthesis and simulation tools).
FM_2_5	FORMALITY	Strength values are ignored by synthesis tools.
FM_2_6A	FORMALITY	Initial statements are ignored by synthesis tools.
FM_2_6B	FORMALITY	Do not use assignment in net/signal declaration.
FM_2_7	FORMALITY	Use named association in port map.
FM_2_8	FORMALITY	Multiple drivers detected for <%item>
FM_2_9	FORMALITY	Using X,Z values or ? is not recommended (differently handled by synthesis and simulation tools).
SYN10_1	IEEE_RTL_ SYNTH_ SUBSET	Writing to global variable <%item> in a function is not supported for synthesis.
SYN10_2	IEEE_RTL_ SYNTH_ SUBSET	Writing to global variable <%item> in a task is not supported for synthesis.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN12_1	IEEE_RTL_ SYNTH_ SUBSET	Macromodules are not supported for synthesis.
SYN12_2	IEEE_RTL_ SYNTH_ SUBSET	Input ports must not be assigned a value.
SYN13_1	IEEE_RTL_ SYNTH_ SUBSET	Specify blocks are ignored.
SYN13_4_1	IEEE_RTL_ SYNTH_ SUBSET	Real literals are not allowed.
SYN14_1	IEEE_RTL_ SYNTH_ SUBSET	System task enables are ignored.
SYN14_1_1	IEEE_RTL_ SYNTH_ SUBSET	Illegal attribute <%item>.
SYN14_2	IEEE_RTL_ SYNTH_ SUBSET	System function calls are not supported for synthesis.
SYN14_3_1	IEEE_RTL_ SYNTH_ SUBSET	Functions in STD.TEXTIO are not supported.
SYN1_1_1_A	IEEE_RTL_ SYNTH_ SUBSET	Process statements are ignored in entities.
SYN1_1_1_B	IEEE_RTL_ SYNTH_ SUBSET	Procedure call statements are ignored in entities.
SYN1_1_1_C	IEEE_RTL_ SYNTH_ SUBSET	Assertion statements are ignored in entities.
SYN1_1_2	IEEE_RTL_ SYNTH_ SUBSET	Port default values are ignored.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN1_1_2_1_A	IEEE_RTL_ SYNTH_ SUBSET	Group declarations are illegal in entities.
SYN1_1_2_1_B	IEEE_RTL_ SYNTH_ SUBSET	Use clauses are illegal in entities.
SYN1_1_2_1_C	IEEE_RTL_ SYNTH_ SUBSET	Disconnection specifications are illegal in entities.
SYN1_1_2_1_D	IEEE_RTL_ SYNTH_ SUBSET	Attribute specifications are illegal in entities.
SYN1_1_2_1_E	IEEE_RTL_ SYNTH_ SUBSET	Signal declarations are illegal in entities.
SYN1_1_2_1_F	IEEE_RTL_ SYNTH_ SUBSET	Attribute declarations are illegal in entities.
SYN1_1_2_1_G	IEEE_RTL_ SYNTH_ SUBSET	Group template declarations are illegal in entities.
SYN1_1_2_1_H	IEEE_RTL_ SYNTH_ SUBSET	Shared variable declarations are illegal in entities.
SYN1_1_2_1_I	IEEE_RTL_ SYNTH_ SUBSET	Constant declarations are illegal in entities.
SYN1_1_2_1_J	IEEE_RTL_ SYNTH_ SUBSET	Subtype declarations are illegal in entities.
SYN1_1_2_1_K	IEEE_RTL_ SYNTH_ SUBSET	Type declarations are illegal in entities.
SYN1_1_2_1_L	IEEE_RTL_ SYNTH_ SUBSET	Subtype declarations are illegal in entities.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN1_1_2_1_M	IEEE_RTL_ SYNTH_ SUBSET	File declarations are illegal in entities.
SYN1_1_2_1_N	IEEE_RTL_ SYNTH_ SUBSET	Alias declarations are illegal in entities.
SYN_1_3	IEEE_RTL_ SYNTH_ SUBSET	Only generics of type integer are accepted.
SYN1_2_1_1_A	IEEE_RTL_ SYNTH_ SUBSET	File declarations are illegal in architectures.
SYN1_2_1_1_B	IEEE_RTL_ SYNTH_ SUBSET	Disconnection specifications are ignored in architectures.
SYN1_2_1_1_C	IEEE_RTL_ SYNTH_ SUBSET	Attribute specifications are ignored in architectures.
SYN1_2_1_1_D	IEEE_RTL_ SYNTH_ SUBSET	Alias declarations are ignored in architectures.
SYN1_2_1_1_E	IEEE_RTL_ SYNTH_ SUBSET	Group declarations are ignored in architectures.
SYN1_2_1_1_F	IEEE_RTL_ SYNTH_ SUBSET	Shared variable declarations are ignored in architectures.
SYN1_2_1_1_G	IEEE_RTL_ SYNTH_ SUBSET	Group template declarations are ignored in architectures.
SYN1_2_1_2	IEEE_RTL_ SYNTH_ SUBSET	Use clauses can only indicate package declarations.
SYN1_3_1_A	IEEE_RTL_ SYNTH_ SUBSET	Group declarations are illegal in configurations declarations.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN1_3_1_B	IEEE_RTL_ SYNTH_ SUBSET	Use clauses are illegal in configurations declarations.
SYN1_3_1_C	IEEE_RTL_ SYNTH_ SUBSET	Attribute specifications are illegal in configurations declarations.
SYN1_3_2	IEEE_RTL_ SYNTH_ SUBSET	Component configurations are illegal in block declarations.
SYN1_3_3	IEEE_RTL_ SYNTH_ SUBSET	Use clauses are illegal in block declarations.
SYN1_3_4_A	IEEE_RTL_ SYNTH_ SUBSET	Block statement labels are illegal in block declarations.
SYN1_3_4_B	IEEE_RTL_ SYNTH_ SUBSET	Generate statement labels are illegal in block declarations.
SYN2_1_1	IEEE_RTL_ SYNTH_ SUBSET	Default values for subprogram parameters are ignored.
SYN2_1_2	IEEE_RTL_ SYNTH_ SUBSET	Impure subprograms are not allowed.
SYN2_1_3	IEEE_RTL_ SYNTH_ SUBSET	Pure keyword cannot be used.
SYN2_2_1_A	IEEE_RTL_ SYNTH_ SUBSET	File declarations are illegal in subprograms.
SYN2_2_1_B	IEEE_RTL_ SYNTH_ SUBSET	Group template declarations are illegal in subprograms.
SYN2_2_1_C	IEEE_RTL_ SYNTH_ SUBSET	Group declarations are illegal in subprograms.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN2_2_1_D	IEEE_RTL_ SYNTH_ SUBSET	Alias declarations are ignored in subprograms.
SYN2_2_2	IEEE_RTL_ SYNTH_ SUBSET	Assertion statements are ignored in subprogram bodies.
SYN2_2_3	IEEE_RTL_ SYNTH_ SUBSET	Report statements are ignored in subprogram bodies.
SYN2_2_4	IEEE_RTL_ SYNTH_ SUBSET	Wait statements are ignored in subprogram bodies.
SYN2_2_5	IEEE_RTL_ SYNTH_ SUBSET	Recursion is illegal unless bounded by a static value.
SYN2_2_6	IEEE_RTL_ SYNTH_ SUBSET	Use clauses can only indicate package declarations.
SYN2_5_1	IEEE_RTL_ SYNTH_ SUBSET	User-defined resolution functions are illegal.
SYN2_5_2_A	IEEE_RTL_ SYNTH_ SUBSET	File declarations are illegal in package declarations.
SYN2_5_2_B	IEEE_RTL_ SYNTH_ SUBSET	Group declarations are illegal in package declarations.
SYN2_5_2_C	IEEE_RTL_ SYNTH_ SUBSET	Alias declarations are ignored in package declarations.
SYN2_5_2_D	IEEE_RTL_ SYNTH_ SUBSET	Disconnection specifications are ignored in package declarations.
SYN2_5_2_E	IEEE_RTL_ SYNTH_ SUBSET	Shared variable declarations are illegal in package declarations.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN2_5_2_F	IEEE_RTL_ SYNTH_ SUBSET	Global signal declarations in package declaration cannot be used.
SYN2_5_2_G	IEEE_RTL_ SYNTH_ SUBSET	Group template declarations are illegal in package declarations.
SYN2_5_3	IEEE_RTL_ SYNTH_ SUBSET	Signal declarations in packages must have default value.
SYN2_5_4	IEEE_RTL_ SYNTH_ SUBSET	Use clauses can only indicate package declarations.
SYN2_6_1_A	IEEE_RTL_ SYNTH_ SUBSET	Group template declarations are illegal in package bodies.
SYN2_6_1_B	IEEE_RTL_ SYNTH_ SUBSET	Shared variable declarations are illegal in package bodies.
SYN2_6_1_C	IEEE_RTL_ SYNTH_ SUBSET	File declarations are illegal in package bodies.
SYN2_6_1_D	IEEE_RTL_ SYNTH_ SUBSET	Group declarations are illegal in package bodies.
SYN2_6_2	IEEE_RTL_ SYNTH_ SUBSET	Alias declarations are ignored in package bodies.
SYN2_6_3	IEEE_RTL_ SYNTH_ SUBSET	Use clauses can only indicate package declarations.
SYN3_1_1	IEEE_RTL_ SYNTH_ SUBSET	Floating type definitions are ignored.
SYN3_1_2	IEEE_RTL_ SYNTH_ SUBSET	Physical type definitions are ignored.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN3_1_3	IEEE_RTL_ SYNTH_ SUBSET	Integer value must be in range $-(2^{31}-1)$ to $(2^{31}-1)$.
SYN3_1_4	IEEE_RTL_ SYNTH_ SUBSET	Null ranges are illegal.
SYN3_1_5	IEEE_RTL_ SYNTH_ SUBSET	Predefined type SEVERITY_LEVEL is ignored.
SYN3_1_6	IEEE_RTL_ SYNTH_ SUBSET	Predefined type STD_FILE_OPEN_KIND is illegal.
SYN3_1_7	IEEE_RTL_ SYNTH_ SUBSET	Predefined type STD_FILE_OPEN_STATUS is illegal.
SYN3_2_10	IEEE_RTL_ SYNTH_ SUBSET	trior nets are not supported for synthesis.
SYN3_2_1_A	IEEE_RTL_ SYNTH_ SUBSET	Multi-dimension arrays are illegal.
SYN3_2_1_B	IEEE_RTL_ SYNTH_ SUBSET	trireg nets are not supported for synthesis.
SYN3_2_2	IEEE_RTL_ SYNTH_ SUBSET	Drive strengths in net declaration are ignored.
SYN3_2_3	IEEE_RTL_ SYNTH_ SUBSET	Charge strengths in net declaration are ignored.
SYN3_2_4	IEEE_RTL_ SYNTH_ SUBSET	Delays in net declaration are ignored.
SYN3_2_5	IEEE_RTL_ SYNTH_ SUBSET	Delays (delay2) in net declaration are ignored.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN3_2_6	IEEE_RTL_ SYNTH_ SUBSET	Delays (delay3) in net declaration are ignored.
SYN3_2_7	IEEE_RTL_ SYNTH_ SUBSET	tri1 nets are not supported for synthesis.
SYN3_2_8	IEEE_RTL_ SYNTH_ SUBSET	triand nets are not supported for synthesis.
SYN3_2_9	IEEE_RTL_ SYNTH_ SUBSET	tri0 nets are not supported for synthesis.
SYN3_3_1	IEEE_RTL_ SYNTH_ SUBSET	Access type definitions are ignored.
SYN3_4_1	IEEE_RTL_ SYNTH_ SUBSET	File type definitions are ignored.
SYN3_9_1	IEEE_RTL_ SYNTH_ SUBSET	Time declarations are not supported for synthesis.
SYN3_9_2	IEEE_RTL_ SYNTH_ SUBSET	Real declarations are not supported for synthesis.
SYN3_9_3	IEEE_RTL_ SYNTH_ SUBSET	Realtime declarations are not supported for synthesis.
SYN4_1_1_A	IEEE_RTL_ SYNTH_ SUBSET	Incomplete type declarations are ignored.
SYN4_1_1_B	IEEE_RTL_ SYNTH_ SUBSET	Expressions of type mintypmax are ignored.
SYN4_1_2	IEEE_RTL_ SYNTH_ SUBSET	The case equality operator '===’ is not supported in binary operations.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN4_1_3	IEEE_RTL_ SYNTH_ SUBSET	The case inequality operator ‘!==' is not supported in binary operations.
SYN4_1_4	IEEE_RTL_ SYNTH_ SUBSET	Real numbers are not supported for synthesis.
SYN4_3_1_1_1	IEEE_RTL_ SYNTH_ SUBSET	Deferred constant declarations are illegal.
SYN4_3_1_2_1	IEEE_RTL_ SYNTH_ SUBSET	Initial values for signal declarations are ignored.
SYN4_3_1_2_2_ A	IEEE_RTL_ SYNTH_ SUBSET	Bus signal kind is ignored.
SYN4_3_1_2_2_ B	IEEE_RTL_ SYNTH_ SUBSET	Register signal kind is ignored.
SYN4_3_1_3_1	IEEE_RTL_ SYNTH_ SUBSET	Initial values for variable declarations are ignored.
SYN4_3_1_3_2	IEEE_RTL_ SYNTH_ SUBSET	Shared variable declarations are illegal.
SYN4_3_1_4_1	IEEE_RTL_ SYNTH_ SUBSET	File declarations are illegal.
SYN4_3_2_1	IEEE_RTL_ SYNTH_ SUBSET	Buffer mode will be transformed to out mode by synthesis tools.
SYN4_3_2_1_1	IEEE_RTL_ SYNTH_ SUBSET	Illegal association element in association list.
SYN4_3_2_1_2	IEEE_RTL_ SYNTH_ SUBSET	Actuals of mode in and out cannot be same object.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN4_3_2_2	IEEE_RTL_ SYNTH_ SUBSET	Linkage mode is illegal in interface declarations.
SYN4_3_2_3	IEEE_RTL_ SYNTH_ SUBSET	Bus keyword is illegal in interface declarations.
SYN4_3_2_4	IEEE_RTL_ SYNTH_ SUBSET	Default expressions are ignored in interface signal declaration.
SYN4_3_2_5	IEEE_RTL_ SYNTH_ SUBSET	Default expressions are ignored in interface variable declaration.
SYN4_3_2_6	IEEE_RTL_ SYNTH_ SUBSET	Interface file declarations are ignored.
SYN4_3_3_1	IEEE_RTL_ SYNTH_ SUBSET	Alias declarations are ignored.
SYN4_4_1	IEEE_RTL_ SYNTH_ SUBSET	User defined attribute declarations are illegal.
SYN4_6_1	IEEE_RTL_ SYNTH_ SUBSET	Group template declarations are illegal.
SYN4_7_1	IEEE_RTL_ SYNTH_ SUBSET	Group declarations are illegal.
SYN5_1_1	IEEE_RTL_ SYNTH_ SUBSET	Others keyword not allowed in attribute specification.
SYN5_1_2	IEEE_RTL_ SYNTH_ SUBSET	All keyword not allowed in attribute specification.
SYN5_2_1	IEEE_RTL_ SYNTH_ SUBSET	Configuration specifications are ignored.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN5_3_1	IEEE_RTL_ SYNTH_ SUBSET	Disconnection specifications are ignored.
SYN6_1_1	IEEE_RTL_ SYNTH_ SUBSET	Do not use assignment in net declaration.
SYN6_1_2	IEEE_RTL_ SYNTH_ SUBSET	Drive strengths in continuous assign statements are ignored.
SYN6_1_3	IEEE_RTL_ SYNTH_ SUBSET	Delay3 values in continuous assign statements are ignored.
SYN6_1_4	IEEE_RTL_ SYNTH_ SUBSET	Delay2 values in continuous assign statements are ignored.
SYN6_1_5	IEEE_RTL_ SYNTH_ SUBSET	Delay values in continuous assign statements are ignored.
SYN6_6_1	IEEE_RTL_ SYNTH_ SUBSET	Illegal attribute name.
SYN6_6_2	IEEE_RTL_ SYNTH_ SUBSET	Expressions in attribute names are illegal.
SYN7_1_1	IEEE_RTL_ SYNTH_ SUBSET	nmos switch instantiations are not supported for synthesis.
SYN7_1_10	IEEE_RTL_ SYNTH_ SUBSET	rtranif switch instantiations are not supported for synthesis.
SYN7_1_11	IEEE_RTL_ SYNTH_ SUBSET	cmos switch instantiations are not supported for synthesis.
SYN7_1_12	IEEE_RTL_ SYNTH_ SUBSET	rcmos switch instantiations are not supported for synthesis.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN7_1_13	IEEE_RTL_ SYNTH_ SUBSET	pull (pullup and pulldown) gate instantiations are not supported for synthesis.
SYN7_1_14	IEEE_RTL_ SYNTH_ SUBSET	Drive strengths in n input gate instantiations are ignored.
SYN7_1_15	IEEE_RTL_ SYNTH_ SUBSET	Drive strengths in n output gate instantiations are ignored.
SYN7_1_16	IEEE_RTL_ SYNTH_ SUBSET	Drive strengths in enable gate instantiations are ignored.
SYN7_1_17	IEEE_RTL_ SYNTH_ SUBSET	Delay2 values in n input gate instantiations are ignored.
SYN7_1_18	IEEE_RTL_ SYNTH_ SUBSET	Delay values in n input gate instantiations are ignored.
SYN7_1_19	IEEE_RTL_ SYNTH_ SUBSET	Delay2 values in n output gate instantiations are ignored.
SYN7_1_2	IEEE_RTL_ SYNTH_ SUBSET	pmos switch instantiations are not supported for synthesis.
SYN7_1_20	IEEE_RTL_ SYNTH_ SUBSET	Delay values in n_output gate instantiations are ignored.
SYN7_1_21	IEEE_RTL_ SYNTH_ SUBSET	Delay3 values in enable gate instantiations are ignored.
SYN7_1_22	IEEE_RTL_ SYNTH_ SUBSET	Delay2 values in enable gate instantiations are ignored.
SYN7_1_23	IEEE_RTL_ SYNTH_ SUBSET	Delay values in enable gate instantiations are ignored.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN7_1_3	IEEE_RTL_ SYNTH_ SUBSET	rnmos switch instantiations are not supported for synthesis.
SYN7_1_4	IEEE_RTL_ SYNTH_ SUBSET	rpmos switch instantiations are not supported for synthesis.
SYN7_1_5	IEEE_RTL_ SYNTH_ SUBSET	tran switch instantiations are not supported for synthesis.
SYN7_1_6	IEEE_RTL_ SYNTH_ SUBSET	rtans switch instantiations are not supported for synthesis.
SYN7_1_7	IEEE_RTL_ SYNTH_ SUBSET	tranif0 switch instantiations are not supported for synthesis.
SYN7_1_8	IEEE_RTL_ SYNTH_ SUBSET	tranif1 switch instantiations are not supported for synthesis.
SYN7_1_9	IEEE_RTL_ SYNTH_ SUBSET	rtranif0 switch instantiations are not supported for synthesis.
SYN7_2_1	IEEE_RTL_ SYNTH_ SUBSET	STD.STANDARD.XNOR operator not allowed.
SYN7_2_2	IEEE_RTL_ SYNTH_ SUBSET	Standard shift operators not allowed.
SYN7_2_6_1_A	IEEE_RTL_ SYNTH_ SUBSET	RHS of operators /,REM and MOD must be static power of 2.
SYN7_2_6_1_B	IEEE_RTL_ SYNTH_ SUBSET	Operators /,REM and MOD must have positive operands.
SYN7_2_6_1_C	IEEE_RTL_ SYNTH_ SUBSET	LHS of operator ** must have static value 2.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN7_2_6_1_D	IEEE_RTL_ SYNTH_ SUBSET	RHS of operator ** must be positive.
SYN7_3_1_1	IEEE_RTL_ SYNTH_ SUBSET	Null literals are illegal.
SYN7_3_2_1_1	IEEE_RTL_ SYNTH_ SUBSET	Record aggregates are illegal.
SYN8_10_1	IEEE_RTL_ SYNTH_ SUBSET	Labels are not allowed in next statements.
SYN8_11_1	IEEE_RTL_ SYNTH_ SUBSET	Labels are not allowed in exit statements.
SYN8_12_1	IEEE_RTL_ SYNTH_ SUBSET	Labels are not allowed in return statements.
SYN8_1_1_A	IEEE_RTL_ SYNTH_ SUBSET	Labels are not allowed in wait statements.
SYN8_1_1_B	IEEE_RTL_ SYNTH_ SUBSET	UDP declarations are not supported for synthesis.
SYN8_1_2	IEEE_RTL_ SYNTH_ SUBSET	Sensitivity clauses are not allowed in wait statements.
SYN8_1_3	IEEE_RTL_ SYNTH_ SUBSET	Illegal condition in wait_ statement.
SYN8_1_4	IEEE_RTL_ SYNTH_ SUBSET	Timeout clauses are ignored in wait statements.
SYN8_2_1	IEEE_RTL_ SYNTH_ SUBSET	Assertion statements are ignored.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN8_2_2	IEEE_RTL_ SYNTH_ SUBSET	Labels on assertion statements are not supported.
SYN8_3_1	IEEE_RTL_ SYNTH_ SUBSET	Report statements are illegal.
SYN8_4_1	IEEE_RTL_ SYNTH_ SUBSET	Multiple waveform elements are not supported.
SYN8_4_1_1	IEEE_RTL_ SYNTH_ SUBSET	Null waveforms are not supported.
SYN8_4_1_2	IEEE_RTL_ SYNTH_ SUBSET	After expressions in waveforms are not supported.
SYN8_4_3	IEEE_RTL_ SYNTH_ SUBSET	Labels on signal assignments statements are not supported.
SYN8_4_4	IEEE_RTL_ SYNTH_ SUBSET	Keyword reject is not supported.
SYN8_4_5	IEEE_RTL_ SYNTH_ SUBSET	Keyword inertial is not supported.
SYN8_4_6	IEEE_RTL_ SYNTH_ SUBSET	Unaffected waveforms are not supported.
SYN8_5_1	IEEE_RTL_ SYNTH_ SUBSET	Labels on variable assignment statements are not supported.
SYN8_6_1_A	IEEE_RTL_ SYNTH_ SUBSET	UDP instantiations are not supported for synthesis.
SYN8_6_1_B	IEEE_RTL_ SYNTH_ SUBSET	Labels are not allowed in procedure call statements.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN8_7_1	IEEE_RTL_ SYNTH_ SUBSET	Labels are not allowed in if statements.
SYN8_8_1	IEEE_RTL_ SYNTH_ SUBSET	Labels are not allowed in case statements.
SYN8_9_1	IEEE_RTL_ SYNTH_ SUBSET	For loops must have globally static bounds.
SYN8_9_2	IEEE_RTL_ SYNTH_ SUBSET	Wait statements cannot appear inside for loops.
SYN8_9_3	IEEE_RTL_ SYNTH_ SUBSET	While loops are not supported.
SYN9_1	IEEE_RTL_ SYNTH_ SUBSET	Illegal always construct: Does not model any combinational logic or sequential logic.
SYN9_10	IEEE_RTL_ SYNTH_ SUBSET	A falling-edge clock expression should be of the form ‘negedge <clock_name>’
SYN9_11	IEEE_RTL_ SYNTH_ SUBSET	Multiple event lists in an always statement are not supported for synthesis.
SYN9_12	IEEE_RTL_ SYNTH_ SUBSET	Polarity mismatch for asynchronous reset/set/load <%context> : use ‘if(<%context>)’.
SYN9_13	IEEE_RTL_ SYNTH_ SUBSET	Polarity mismatch for asynchronous reset/set/load <%context> : use ‘if(<%context>)’ , ‘if(~<%context>)’ or ‘if(<%context>==1'b0)’.
SYN9_14	IEEE_RTL_ SYNTH_ SUBSET	Level sensitive events are not allowed in a sequential always block.
SYN9_15	IEEE_RTL_ SYNTH_ SUBSET	An asynchronous sequential always block must have one clock signal exactly. <%value> clocks have been detected.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN9_16	IEEE_RTL_ SYNTH_ SUBSET	Initial constructs are ignored.
SYN9_17	IEEE_RTL_ SYNTH_ SUBSET	Procedural continuous assign statements are not supported for synthesis.
SYN9_18	IEEE_RTL_ SYNTH_ SUBSET	Procedural continuous deassign statements are not supported for synthesis.
SYN9_19	IEEE_RTL_ SYNTH_ SUBSET	Procedural continuous force statements are not supported for synthesis.
SYN9_1_1	IEEE_RTL_ SYNTH_ SUBSET	Guard expressions not allowed in block statements.
SYN9_1_2_A	IEEE_RTL_ SYNTH_ SUBSET	Port block headers are not supported.
SYN9_1_2_B	IEEE_RTL_ SYNTH_ SUBSET	Generic block headers are not supported.
SYN9_1_3_A	IEEE_RTL_ SYNTH_ SUBSET	Alias declarations are ignored in block statements.
SYN9_1_3_B	IEEE_RTL_ SYNTH_ SUBSET	Disconnection specifications are ignored in block statements.
SYN9_1_3_C	IEEE_RTL_ SYNTH_ SUBSET	Group template declarations are not supported in block statements.
SYN9_1_3_D	IEEE_RTL_ SYNTH_ SUBSET	File declarations are not supported in block statements.
SYN9_1_3_E	IEEE_RTL_ SYNTH_ SUBSET	Configuration specifications are not supported in block statements.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN9_1_3_F	IEEE_RTL_ SYNTH_ SUBSET	Group declarations are not supported in block statements.
SYN9_1_3_G	IEEE_RTL_ SYNTH_ SUBSET	Shared variable declarations are not supported in block statements.
SYN9_2	IEEE_RTL_ SYNTH_ SUBSET	Missing or redundant signal <%item> in the sensitivity list of an always block.
SYN9_20	IEEE_RTL_ SYNTH_ SUBSET	Procedural continuous release statements are not supported for synthesis.
SYN9_21	IEEE_RTL_ SYNTH_ SUBSET	Repeat event controls in timing control statements are not supported for synthesis.
SYN9_22	IEEE_RTL_ SYNTH_ SUBSET	Delay values are ignored in synthesis.
SYN9_23	IEEE_RTL_ SYNTH_ SUBSET	Forever loop statements are not supported for synthesis.
SYN9_24	IEEE_RTL_ SYNTH_ SUBSET	Repeat loop statements are not supported for synthesis.
SYN9_25	IEEE_RTL_ SYNTH_ SUBSET	While loop statements are not supported for synthesis.
SYN9_26	IEEE_RTL_ SYNTH_ SUBSET	Expression bound in for loop statements should be statically computable.
SYN9_27	IEEE_RTL_ SYNTH_ SUBSET	Initial reg assignment bound in for loop statements should be statically computable.
SYN9_28	IEEE_RTL_ SYNTH_ SUBSET	Wait statements are not supported for synthesis.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN9_29	IEEE_RTL_ SYNTH_ SUBSET	Event triggers are not supported for synthesis.
SYN9_2_1	IEEE_RTL_ SYNTH_ SUBSET	Postponed processes are not supported.
SYN9_2_2_A	IEEE_RTL_ SYNTH_ SUBSET	Group template declarations are not supported in process statements.
SYN9_2_2_B	IEEE_RTL_ SYNTH_ SUBSET	Group declarations are not supported in process statements.
SYN9_2_2_C	IEEE_RTL_ SYNTH_ SUBSET	Alias declarations are ignored in process statements.
SYN9_2_2_D	IEEE_RTL_ SYNTH_ SUBSET	Use clauses in process statements can only refer to package declarations.
SYN9_2_2_E	IEEE_RTL_ SYNTH_ SUBSET	File declarations are not supported in process statements.
SYN9_2_3	IEEE_RTL_ SYNTH_ SUBSET	Variable is read first on at least one flow of control or is read without being initialized within the process body.
SYN9_2_4	IEEE_RTL_ SYNTH_ SUBSET	Only one clock expression per process is allowed.
SYN9_3	IEEE_RTL_ SYNTH_ SUBSET	Do not mix blocking and non-blocking assignments in a combinational always block.
SYN9_30	IEEE_RTL_ SYNTH_ SUBSET	Fork-join block are not supported for synthesis.
SYN9_31	IEEE_RTL_ SYNTH_ SUBSET	Event declarations are not supported for synthesis.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN9_32	IEEE_RTL_ SYNTH_ SUBSET	The always statement must be followed by an event control (@)
SYN9_3_1	IEEE_RTL_ SYNTH_ SUBSET	Postponed concurrent procedure calls are not supported.
SYN9_4	IEEE_RTL_ SYNTH_ SUBSET	Do not use blocking assignments for variables modeling level-sensitive storage devices (latches).
SYN9_4_1	IEEE_RTL_ SYNTH_ SUBSET	Postponed concurrent assertion statements are not supported.
SYN9_4_2	IEEE_RTL_ SYNTH_ SUBSET	Concurrent assertion statements are ignored.
SYN9_5	IEEE_RTL_ SYNTH_ SUBSET	A level-sensitive storage device (latch) may be inferred for <%item>.
SYN9_5_1_1	IEEE_RTL_ SYNTH_ SUBSET	Postponed conditional signal assignments are not supported.
SYN9_5_1_2	IEEE_RTL_ SYNTH_ SUBSET	Illegal conditional waveform.
SYN9_5_1_3_A	IEEE_RTL_ SYNTH_ SUBSET	Inertial keyword on conditional signal assignments is ignored.
SYN9_5_1_3_B	IEEE_RTL_ SYNTH_ SUBSET	Reject expressions on conditional signal assignments is ignored.
SYN9_5_1_3_C	IEEE_RTL_ SYNTH_ SUBSET	Transport keyword on conditional signal assignments is ignored.
SYN9_5_1_3_D	IEEE_RTL_ SYNTH_ SUBSET	Guarded keyword on conditional signal assignments is ignored.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN9_5_1_4	IEEE_RTL_ SYNTH_ SUBSET	Target signal cannot also be a source in conditional signal assignment.
SYN9_5_2_1	IEEE_RTL_ SYNTH_ SUBSET	Postponed selected signal assignments are not supported.
SYN9_5_2_2	IEEE_RTL_ SYNTH_ SUBSET	Illegal selection waveform.
SYN9_5_2_3_A	IEEE_RTL_ SYNTH_ SUBSET	Transport keyword on selected signal assignments is ignored.
SYN9_5_2_3_B	IEEE_RTL_ SYNTH_ SUBSET	Inertial keyword on selected signal assignments is ignored.
SYN9_5_2_3_C	IEEE_RTL_ SYNTH_ SUBSET	Reject expressions on selected signal assignments is ignored.
SYN9_5_2_3_D	IEEE_RTL_ SYNTH_ SUBSET	Guarded keyword on selected signal assignments is ignored.
SYN9_5_2_4	IEEE_RTL_ SYNTH_ SUBSET	Target signal cannot also be a source in selected signal assignment.
SYN9_6	IEEE_RTL_ SYNTH_ SUBSET	A sequential always block must have one clock signal exactly. <% value> clocks have been detected.
SYN9_6_1	IEEE_RTL_ SYNTH_ SUBSET	Entity names are not supported in component instantiation statements.
SYN9_6_2	IEEE_RTL_ SYNTH_ SUBSET	Configuration names are not supported in component instantiation statements.
SYN9_7	IEEE_RTL_ SYNTH_ SUBSET	Do not use blocking assignments for variables modeling edge-sensitive storage devices (flip-flops).

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SYN9_7_1	IEEE_RTL_ SYNTH_ SUBSET	Block declarative part in generate statement is not supported.
SYN9_8	IEEE_RTL_ SYNTH_ SUBSET	Only one edge event should be present in the event list of a synchronous always block.
SYN9_9	IEEE_RTL_ SYNTH_ SUBSET	A clock expression should be of the form 'posedge <clock_name>'.
B_1000	LEDA	Module/unit without I/Os.
B_1001	LEDA	Reading from output port <%item>.
B_1002	LEDA	Port declaration <%item> is unused or partially unused.
B_1005	LEDA	No bidirectional port allowed.
B_1006	LEDA	Tristates are only allowed in specified modules/units.
B_1007	LEDA	Tristate port detected.
B_1008	LEDA	Tristate signal detected.
B_1009	LEDA	Tristate output detected.
B_1010	LEDA	Feedthrough detected for port <%context>.
B_1011	LEDA	Module instantiations is not fully bound. Port <%format> is not completely connected.
B_1013	LEDA	Signal <%context> should not drive multiple ports.
B_1200	LEDA	Nested event control in a task.
B_1201	LEDA	Multiple event control statement in a task.
B_1202	LEDA	<%value> clocks in this unit detected.
B_1204	LEDA	Multi-bit expression used as clock.
B_1205	LEDA	The clock signal <%item> is not coming directly from a port of the current unit.
B_1206	LEDA	Do not use event definitions for clocks.
B_1400	LEDA	Asynchronous reset/set/load signal <%item> is not a primary input to the current unit.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
B_1401	LEDA	Synchronous reset/set/load signal <%item> is not a primary input to the current unit.
B_1402	LEDA	Do not use event definitions for asynchronous resets/sets/loads.
B_1403	LEDA	Flip-flop assigned but not initialized.
B_1405	LEDA	<%value> asynchronous resets in this unit detected.
B_1406	LEDA	<%value> synchronous resets in this unit detected.
B_1409	LEDA	<%value> asynchronous resets in always/process block.
B_1410	LEDA	<%value> synchronous resets in always/process block.
B_1411	LEDA	<%value> asynchronous sets in this unit detected.
B_1412	LEDA	<%value> synchronous sets in this unit detected.
B_1413	LEDA	<%value> asynchronous sets in always/process block.
B_1414	LEDA	<%value> synchronous sets in always/process block.
B_1415	LEDA	<%value> asynchronous loads in this unit detected.
B_1416	LEDA	<%value> synchronous loads in this unit detected.
B_1417	LEDA	<%value> asynchronous loads in always/process block.
B_1418	LEDA	<%value> synchronous loads in always/process block.
B_2000	LEDA	System tasks are not allowed.
B_2001	LEDA	Shift by a non constant value is not allowed.
B_2002	LEDA	Disable statement in always construct may not be synthesizable.
B_2003	LEDA	Disable statement in task may not be synthesizable.
B_2004	LEDA	Disable statement in function may not be synthesizable.
B_2005	LEDA	Function of type real are not synthesizable.
B_2006	LEDA	Event declarations are not allowed.
B_2007	LEDA	Same operands on both sides of assignment detected.
B_2008	LEDA	Delays in signal assignment are ignored by synthesis tool.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
B_2009	LEDA	Delays in conditional signal assignment are ignored by synthesis tool.
B_2010	LEDA	Non synthesizable operator === !== encountered.
B_2010	LEDA	Variable is not always initialized in process body before being read.
B_3001	LEDA	Array of integer is not allowed.
B_3002	LEDA	Array of time is not allowed.
B_3003	LEDA	Test expression in if_statement is expected to be one bit wide.
B_3004_A	LEDA	Unrecommended blocking assignment (converting integer to real).
B_3005_A	LEDA	Unrecommended blocking assignment (converting unsigned to real).
B_3005_B	LEDA	Unrecommended non blocking assignment (converting unsigned to real).
B_3006_A	LEDA	Unrecommended blocking assignment (converting real to integer).
B_3007_A	LEDA	Unrecommended blocking assignment (converting unsigned to integer).
B_3008_A	LEDA	Unrecommended blocking assignment (converting integer to unsigned).
B_3009_A	LEDA	Unrecommended blocking assignment (converting real to unsigned).
B_3010	LEDA	Loop index must be declared as integer.
B_3200	LEDA	Unequal length operand in bit/arithmetic operator.
B_3201	LEDA	Unequal length operand in comparison operator.
B_3202	LEDA	Delay is not constant expression.
B_3203	LEDA	The expression in for loop must not be constant.
B_3204	LEDA	? in based number constant is not allowed.
B_3206	LEDA	X in based number constant.
B_3207	LEDA	Z in based number constant.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
B_3208	LEDA	Unequal length in LHS and RHS in assignment.
B_3209	LEDA	Unequal length port and connection in module instantiation.
B_3210	LEDA	Unequal length arguments in function call or task enable.
B_3211	LEDA	Unequal length between case expression and case item condition in case, casex or casez.
B_3400	LEDA	Empty block found: No statements in block.
B_3401	LEDA	Blocking delay not allowed in non-blocking assignment.
B_3402	LEDA	Task assigns global variable <%item>.
B_3407	LEDA	No null statements in process statement.
B_3408	LEDA	Case condition expression should not be a constant.
B_3409	LEDA	While condition expression is constant.
B_3410	LEDA	X in case expression.
B_3411	LEDA	Assignment to a supply0 type net.
B_3412	LEDA	Assignment to a supply1 type net.
B_3413	LEDA	Task call in a combinational block.
B_3414	LEDA	Task call in a sequential block.
B_3415	LEDA	<%context> has no drivers. It should have at least one.
B_3416	LEDA	Use blocking assignments in combinatorial block.
B_3417	LEDA	Use non-blocking assignments in sequential block.
B_3418	LEDA	Redundant signal <%item> in sensitivity list.
B_3419	LEDA	Missing signal <%item> in sensitivity list.
B_3601	LEDA	<%value> blocks used to code state machine. Two blocks should be used.
B_3602	LEDA	Moore style description of state machine is recommended.
B_3604	LEDA	Assign a default state to the state machines.
B_3605_A	LEDA	Use parameter declarations to define the state vector of a state machine.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
B_3605_B	LEDA	Use an enumerated type to define the state vector of a state machine.
B_3607	LEDA	The number of states in a state machine should be a power of 2.
B_3608	LEDA	The number of states in a state machine should be less than 40.
B_3609	LEDA	In state machine, keep FSM logic and non-FSM logic apart.
B_4001	LEDA	Process blocks should have a label.
B_4002	LEDA	Comments are required on preceding line of function or procedure declaration.
B_5000	LEDA	No non-blocking assignments in always_comb.
B_5001	LEDA	No non-blocking assignments in always_latch.
B_5005	LEDA	No latches or flip-flops in always_comb.
B_5006	LEDA	No flip-flops in always_latch.
B_5007	LEDA	No event controls or delays in always_comb.
B_5008	LEDA	No event controls or delays in always_latch.
B_5009	LEDA	Only one event control in always_ff.
B_5010	LEDA	There must be at least one latch in always_latch.
B_5011	LEDA	There must be at least one flip-flop in always_ff.
B_5012	LEDA	always_comb, always_latch, always_ff statements used - a SystemVerilog feature.
B_5015	LEDA	Do-while loop used - a SystemVerilog feature.
B_5016	LEDA	unique/priority keywords used for conditional and case statements not allowed - a SystemVerilog feature.
B_5018	LEDA	Scope/lifetime specified for functions & tasks - a SystemVerilog feature.
B_5021	LEDA	Assignment expression in event control - a SystemVerilog feature.
B_5022	LEDA	Continuous assign used for variables - a SystemVerilog feature.
B_5025	LEDA	Process statement used - a SystemVerilog feature.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
B_5027	LEDATA	Array querying functions used - a SystemVerilog feature.
B_5028	LEDATA	Input mode - a default mode for task/function argument direction.
B_5101	LEDATA	Enumerated data types used - a SystemVerilog feature.
B_5102	LEDATA	Constant data type used - a SystemVerilog feature.
B_5103	LEDATA	Structure data type used - a SystemVerilog feature.
B_5104	LEDATA	Union data type used - a SystemVerilog feature.
B_5105	LEDATA	4-state logic data type used - a SystemVerilog feature.
B_5106	LEDATA	Integer data type used - a SystemVerilog feature.
B_5107	LEDATA	Casting used - a SystemVerilog feature.
B_5108	LEDATA	void data types used - a SystemVerilog feature.
B_5109	LEDATA	User defined data types used - a SystemVerilog feature.
B_5200	LEDATA	Interface used - a SystemVerilog feature.
B_5201	LEDATA	Interface ports used - a SystemVerilog feature.
B_5202	LEDATA	modports in interface declaration detected.
B_5203	LEDATA	Tasks and functions in mod ports detected.
B_5204	LEDATA	Export & Import of tasks and functions in interfaces detected.
B_5205	LEDATA	No modport interface.
B_5206	LEDATA	Nested module found - a SystemVerilog feature.
B_5207	LEDATA	Implicit name port connections used - a SystemVerilog feature.
B_5208	LEDATA	Implicit .* port connections used - a SystemVerilog feature.
C_1000	LEDATA	Asynchronous feedback loop detected.
C_1001	LEDATA	Flip-flop with fixed value data input is detected.
C_1002	LEDATA	Latch with fixed value data input is detected.
C_1003	LEDATA	Latch detected in design (inferred or instantiated).
C_1004	LEDATA	Glue logic at top-level is detected.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
C_1005	LEDA	Top-level outputs are not registered.
C_1006	LEDA	Top-level inputs are not registered.
C_1007	LEDA	Pulse generator detected.
C_1009	LEDA	Multiple non-tristate drivers to signal <%item> detected.
C_1200	LEDA	Only one clock allowed in the design. <%value> clocks have been identified.
C_1201	LEDA	Clocks must not be used as data.
C_1202	LEDA	Data must be registered by 2 flip-flops when changing clock domain.
C_1203	LEDA	Internally generated clock detected. (chip level).
C_1204	LEDA	No gated clock except in clock generator CKGEN.
C_1208	LEDA	Multiplexed clock is detected.
C_1209	LEDA	Register with fixed value clock is detected.
C_1400	LEDA	Only 1 reset/set/load allowed in the design. <%value> have been detected.
C_1401	LEDA	Avoid gated resets/sets/loads in design.
C_1402	LEDA	No gated reset/set/load except in reset/set/load generator RSTGEN.
C_1403	LEDA	Buffers must not be explicitly added to reset/set/load paths.
C_1404	LEDA	Signal is used both as synchronous and asynchronous reset/set/load.
C_1405	LEDA	Register with fixed value reset/set/load is detected.
C_1406	LEDA	Register with no reset/set/load is detected.
G_5210_2	RMM_RTL_CODING_GUIDELINES	Declare one port per line.
G_5214_2	RMM_RTL_CODING_GUIDELINES	Use vector operations on arrays rather than loops.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
G_521_11	RMM_RTL_CODING_GUIDELINES	Use same name/similar names for ports (<%formal> and signals (<actual>.
G_523_1_D	RMM_RTL_CODING_GUIDELINES	Port assignments are not allowed in testbench architectures.
G_531_2	RMM_RTL_CODING_GUIDELINES	Use std_logic than std_ulogic when possible.
G_531_4	RMM_RTL_CODING_GUIDELINES	Types bit and bit_vector should not be used.
G_532_1	RMM_RTL_CODING_GUIDELINES	Do not use literals in statements, use constants instead.
G_533_1	RMM_RTL_CODING_GUIDELINES	All definitions for a design should be in a separate package.
G_536_2	RMM_RTL_CODING_GUIDELINES	Do not instantiate verilog predefined gate <%item> in the design.
G_537_1	RMM_RTL_CODING_GUIDELINES	Generate statements are not allowed.
G_537_2	RMM_RTL_CODING_GUIDELINES	Block statements are not allowed.
G_537_3	RMM_RTL_CODING_GUIDELINES	Do not use complex expressions to initialize constants.
G_541_1	RMM_RTL_CODING_GUIDELINES	Avoid using both positive-edge and negative-edge triggered flip-flops in your design.
G_542_1	RMM_RTL_CODING_GUIDELINES	Buffers should not be explicitly added to clock path.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
G_543_1	RMM_RTL_ CODING_ GUIDELINES	Gated clocks are not allowed in the design.
G_544_1	RMM_RTL_ CODING_ GUIDELINES	Clocks should be visible from top unit.
G_546_1	RMM_RTL_ CODING_ GUIDELINES	Avoid internally generated reset/set/load <%item>.
G_551_1_B	RMM_RTL_ CODING_ GUIDELINES	The always keyword must be followed by an event list @(...) in a sequential block.
G_551_1_C	RMM_RTL_ CODING_ GUIDELINES	Use 'if (%context> == 'b0)' or 'if (%context> == 'b1)' for synchronous reset/set/load expressions: <%context>.
G_551_1_D	RMM_RTL_ CODING_ GUIDELINES	Use 'if (%context> == 'b1)' for rising edge asynchronous reset/set/load expressions.
G_551_1_E	RMM_RTL_ CODING_ GUIDELINES	Use 'if (%context> == 'b0)' for falling edge asynchronous reset/set/load expressions.
G_551_1_F	RMM_RTL_ CODING_ GUIDELINES	Use if (<%item> = '1') or if(<%item> = '0') for reset/set/load expressions.
G_551_1_G	RMM_RTL_ CODING_ GUIDELINES	Do not use initial constructs to initialize signals.
G_551_1_H	RMM_RTL_ CODING_ GUIDELINES	There should be exactly one clock signal in the sensitivity list of a sequential block. <%value> clocks have been detected.
G_551_1_I	RMM_RTL_ CODING_ GUIDELINES	There should be at most one asynchronous reset/set/load signal in a sequential block.
G_551_1_J	RMM_RTL_ CODING_ GUIDELINES	An asynchronous reset/set/load signal should be preceded by the keyword 'posedge or 'negedge in the sensitivity list of a sequential block.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
G_551_1_K	RMM_RTL_ CODING_ GUIDELINES	There should be at most one synchronous reset/set/load signal in a sequential block.
G_551_1_L	RMM_RTL_ CODING_ GUIDELINES	Always block with event and level expression detected in sensitivity list. This block is not synthesizable.
G_553_1	RMM_RTL_ CODING_ GUIDELINES	Avoid asynchronous feedback loops.
G_556_1	RMM_RTL_ CODING_ GUIDELINES	Use signals instead of variables (suitable for synthesis).
G_559_1	RMM_RTL_ CODING_ GUIDELINES	<% value> blocks used to code state machine. Two block should be used.
G_559_2_A	RMM_RTL_ CODING_ GUIDELINES	Use parameter statements to define the state vector of a state machine.
G_559_2_B	RMM_RTL_ CODING_ GUIDELINES	Create an enumerated type to define the state vector of a state machine.
G_559_3	RMM_RTL_ CODING_ GUIDELINES	In state machine, keep FSM logic and non-FSM logic apart.
G_559_4	RMM_RTL_ CODING_ GUIDELINES	Assign a default state to the state machine.
G_561_1	RMM_RTL_ CODING_ GUIDELINES	Drivers of output ports should be registered: %s
G_564_1	RMM_RTL_ CODING_ GUIDELINES	Avoid using asynchronous logic.
G_568_1	RMM_RTL_ CODING_ GUIDELINES	Avoid glue logic at top level.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
R_5210_1_A	RMM_RTL_CODING_GUIDELINES	Ports must be declared in the following order: in out inout buffer linkage.
R_5210_1_B	RMM_RTL_CODING_GUIDELINES	Ports must be declared in the following order: input inout output.
R_5211_1	RMM_RTL_CODING_GUIDELINES	Use named association when instantiating design units.
R_5215_1	RMM_RTL_CODING_GUIDELINES	Every process must have a (meaningful) process label.
R_521_10	RMM_RTL_CODING_GUIDELINES	Always use descending range for multi-bit signals and ports.
R_522_1	RMM_RTL_CODING_GUIDELINES	Underscores are not allowed in top level port names.
R_522_2	RMM_RTL_CODING_GUIDELINES	Linkage mode is not allowed for top level port declarations.
R_522_3	RMM_RTL_CODING_GUIDELINES	Top level port must be of type std_logic(_vector), signed or unsigned.
R_524_1_A	RMM_RTL_CODING_GUIDELINES	Header comments are missing.
R_524_1_B	RMM_RTL_CODING_GUIDELINES	Modification field missing from header comment.
R_524_1_C	RMM_RTL_CODING_GUIDELINES	Description field missing from header comment.
R_524_1_D	RMM_RTL_CODING_GUIDELINES	Date field missing from header comment.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
R_524_1_E	RMM_RTL_ CODING_ GUIDELINES	Author field missing from header comment.
R_524_1_F	RMM_RTL_ CODING_ GUIDELINES	File name field missing from header comment.
R_525_1_B	RMM_RTL_ CODING_ GUIDELINES	Function must have a header comment.
R_525_1_C	RMM_RTL_ CODING_ GUIDELINES	Task must have a header comment.
R_525_1_D	RMM_RTL_ CODING_ GUIDELINES	Process must have a header comment.
R_525_1_E	RMM_RTL_ CODING_ GUIDELINES	Subprogram must have a header comment.
R_526_1	RMM_RTL_ CODING_ GUIDELINES	Use a separate line for each HDL statement.
R_529_1	RMM_RTL_ CODING_ GUIDELINES	VHDL or Verilog reserved words cannot be used as identifiers.
R_531_1	RMM_RTL_ CODING_ GUIDELINES	All types and subtypes should be based on IEEE standard types.
R_552_1	RMM_RTL_ CODING_ GUIDELINES	Latch inferred for <%item>.
R_554_1_A	RMM_RTL_ CODING_ GUIDELINES	Redundant signal <%item> in sensitivity list.
R_554_1_B	RMM_RTL_ CODING_ GUIDELINES	Missing signal <%item> in sensitivity list.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
R_555_1_A	RMM_RTL_CODING_GUIDELINES	Use non-blocking assignments in sequential always blocks.
R_555_1_B	RMM_RTL_CODING_GUIDELINES	Use blocking assignments in combinational always blocks.
SC_001	SCIROCCO_CYCLE	Only access types of type LINE, from package textio, are supported in cycle mode.
SC_002	SCIROCCO_CYCLE	Generics of records type are not supported in cycle mode.
SC_003	SCIROCCO_CYCLE	Enumeration values may not be used as for loop bounds in cycle mode.
SC_004	SCIROCCO_CYCLE	Enumeration values may not be used as for-generate loop bounds in cycle mode.
SC_006	SCIROCCO_CYCLE	Deferred constant declarations are not supported in cycle mode.
SC_009	SCIROCCO_CYCLE	Incomplete type declarations are not supported in cycle mode.
SC_010	SCIROCCO_CYCLE	Shared variable declarations are not supported in cycle mode.
SC_011	SCIROCCO_CYCLE	Only the USE clause is allowed in the declarative part of configuration in cycle mode.
SC_012	SCIROCCO_CYCLE	Postponed processes are not supported in cycle mode.
SC_016	SCIROCCO_CYCLE	Multiple waveform elements (fragments) not supported in a signal assignment in cycle mode.
SC_018	SCIROCCO_CYCLE	Signal declaration in a generate statement is not supported in cycle mode.
SC_019	SCIROCCO_CYCLE	Pulse rejection limit expression is not supported in cycle mode.
SC_020	SCIROCCO_CYCLE	Extended identifiers are not supported in cycle mode.
SC_022	SCIROCCO_CYCLE	Subelement association for record formals is not supported in cycle mode.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SC_023	SCIROCCO_CYCLE	Type TIME is the only supported physical type in cycle mode.
SC_026	SCIROCCO_CYCLE	Record declaration inside the subprogram is not supported in cycle mode.
SC_027	SCIROCCO_CYCLE	Signal declaration in a package (global signal) is not supported in cycle mode.
SC_028	SCIROCCO_CYCLE	File declaration in a package is not supported in cycle mode.
SC_029	SCIROCCO_CYCLE	Allocators are not supported in cycle mode.
SC_030	SCIROCCO_CYCLE	'X' checking not allowed in 2-state cycle mode (allowed in 4-state cycle mode).
SC_031	SCIROCCO_CYCLE	'Z' checking not allowed in 2-state cycle mode (allowed in 4-state cycle mode).
SC_032	SCIROCCO_CYCLE	Guard conditions for blocks are not supported in cycle mode.
SC_100	SCIROCCO_CYCLE	Bus signal kind is ignored in cycle mode.
SC_101	SCIROCCO_CYCLE	Register signal kind is ignored in cycle mode.
SC_102	SCIROCCO_CYCLE	In assignment using transport clause, delays are ignored in cycle mode.
SC_103	SCIROCCO_CYCLE	AFTER clause - delays are ignored in cycle mode.
SC_104	SCIROCCO_CYCLE	DISCONNECT specification is ignored in cycle mode.
SC_105	SCIROCCO_CYCLE	Statements in an entity are ignored in cycle mode.
SC_107	SCIROCCO_CYCLE	Missing or redundant signals in the process sensitivity list. Signal <%item> is missing or redundant.
SC_201	SCIROCCO_CYCLE	Variable assignment outside pure sequential region - extra cycle mode trigger.
SC_202	SCIROCCO_CYCLE	Variable is read first on at least one flow of control - extra cycle mode trigger.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SC_204	SCIROCCO_CYCLE	Use of both rising edge and falling edge triggered logic will yield extra cycle mode triggers which in turn will degrade simulation performance.
SC_300	SCIROCCO_CYCLE	Gated clocks create extra cycle mode triggers.
SC_301	SCIROCCO_CYCLE	In any cycle mode partitioned block, the clock should be an input to the block.
SC_302	SCIROCCO_CYCLE	In any cycle mode partitioned block, the asynchronous reset/set/load should be an input to the block; <%item> is not an input.
SC_303	SCIROCCO_CYCLE	<%value> clocks have been detected in this cycle mode partitioned block.
SC_304	SCIROCCO_CYCLE	<%value> resets have been detected in this cycle mode partitioned block.
SC_305	SCIROCCO_CYCLE	Asynchronous feedback loops are not recommended in cycle mode.
SC_306	SCIROCCO_CYCLE	<%value> sets have been detected in this cycle mode partitioned block.
SC_307	SCIROCCO_CYCLE	<%value> loads have been detected in this cycle mode partitioned block.
VCS_1	VCS	Avoid asynchronous feedback loops.
VCS_10	VCS	Do not implicit wire declaration
VCS_11	VCS	Implicit wire declaration is not supported.
VCS_12	VCS	Use only non-blocking assignments without delays in always block.
VCS_14	VCS	Only non-blocking assignments allowed in synchronous blocks.
VCS_15	VCS	Only blocking assignments allowed in combinational blocks.
VCS_17	VCS	Regs must be assigned by one block only. Multiple drivers detected for <%item>.
VCS_2_2	VCS	Avoid using time declarations.
VCS_2_3	VCS	Avoid using event triggers.
VCS_2_4	VCS	Avoid using triereg.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
VCS_2_5	VCS	Avoid using ranges/arrays for integers.
VCS_3	VCS	Avoid using n_output_gate.
VCS_31	VCS	Static data types are not supported in \$root.
VCS_33	VCS	unique/priority not supported in conditional statement.
VCS_34	VCS	unique/priority not supported in case/casex/casez statement.
VCS_3_1	VCS	Avoid using n_input_gate.
VCS_3_2	VCS	Avoid using enable_gate.
VCS_3_3	VCS	Avoid using mos_switch.
VCS_3_4	VCS	Avoid using pass_switch.
VCS_3_5	VCS	Avoid using pass enable switch.
VCS_3_6	VCS	Avoid using cmos_switch.
VCS_3_7	VCS	Avoid using pull_gate.
VCS_4	VCS	Avoid declaring strengths with continuous assignments.
VCS_42	VCS	Casting is not supported.
VCS_45	VCS	Generic interface ports not supported.
VCS_5	VCS	Missing or redundant signals in the sensitivity list of a combinational block. Signal <%item> is missing or redundant.
VCS_53	VCS	Import/export of tasks and functions is not supported.
VCS_54	VCS	Process statement is not supported.
VCS_55	VCS	Nested module/interface declaration is not supported.
VCS_7	VCS	Avoid using case statement in sequential blocks.
VCS_7_1	VCS	Avoid using repeat in always blocks.
VCS_7_10	VCS	Avoid task enable in always blocks.
VCS_7_11	VCS	Avoid disable statement in always construct.
VCS_7_3	VCS	Avoid using wait statement in always blocks.
VCS_7_4	VCS	Avoid using fork-join in always block.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
VCS_7_5	VCS	Avoid assign statements in always blocks.
VCS_7_6	VCS	Avoid deassign in always blocks.
VCS_7_7	VCS	Avoid force in always blocks.
VCS_7_9	VCS	Avoid release in always blocks.
VCS_8	VCS	Avoid procedural assignments using a variable to bit-select on LHS.
VCS_9	VCS	Missing or redundant signals in the sensitivity list of a sequential block. Signal <%item> is missing or redundant.
E25	VERILINT	Bits are backwards.
E267	VERILINT	Range index out of bound.
E268	VERILINT	Index out of bound.
E304	VERILINT	Drive strength cannot be given to a net.
E368	VERILINT	Variable <%item> previously declared as a vector.
E54	VERILINT	Instance name required for module.
E66	VERILINT	Not a constant expression.
W110	VERILINT	Incompatible width.
W112	VERILINT	Nested event control construct.
W122	VERILINT	Variable <%item> is not in the sensitivity list.
W126	VERILINT	Non integer delay.
W127	VERILINT	Delay has X or Z.
W129	VERILINT	Delay is not a constant.
W131	VERILINT	Potential loss of precision in multiplication.
W154	VERILINT	Implicit wire declaration.
W159	VERILINT	Constant condition expression.
W161	VERILINT	Constant expression in conditional select.
W163	VERILINT	Truncation of bits in constant. Most significant bits are lost.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
W182	VERILINT	Illegal statement for synthesis.
W187	VERILINT	Default clause is not the last clause in case statement.
W188	VERILINT	Destination variable is input.
W192	VERILINT	Empty block.
W20	VERILINT	Assign statement may not be synthesizable.
W21	VERILINT	Deassign statement may not be synthesizable.
W215	VERILINT	Bit select for integer or time variable.
W216	VERILINT	Range select for integer or time variable.
W224	VERILINT	Multi-bit expression when one bit expression is expected.
W225	VERILINT	Case item expression is not constant.
W226	VERILINT	Case-select expression is constant.
W228	VERILINT	While condition expression is constant.
W244	VERILINT	Shift by non-constant.
W250	VERILINT	Disable statement is not synthesizable.
W257	VERILINT	Delays ignored by synthesis tools.
W263	VERILINT	Case expression out of range.
W280	VERILINT	Delay in non blocking assignment.
W287	VERILINT	Unconnected port <%formal>.
W289	VERILINT	Multiply connected port.
W294	VERILINT	Unsynthesizable real variable <%item>.
W299	VERILINT	Blocking repeat assignment.
W300	VERILINT	Non-blocking repeat assignment.
W306	VERILINT	Converting integer to real.
W307	VERILINT	Converting unsigned to real.
W308	VERILINT	Converting real to integer.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
W311	VERILINT	Converting real to unsigned.
W312	VERILINT	Converting real to single bit (logical).
W313	VERILINT	Converting integer to single bit (logical).
W314	VERILINT	Converting vector (unsigned) to single bit (logical).
W322	VERILINT	Multiple event control statement.
W332	VERILINT	Not all possible cases covered by default case exists.
W335	VERILINT	Non blocking delay assignment in combinational always block.
W336	VERILINT	Blocking assignment. In sequential always blocks consider using non-blocking assignment.
W337	VERILINT	Real comparison in case item.
W339	VERILINT	Non synthesizable operator.
W341	VERILINT	Extension of zero bits in a constant.
W342	VERILINT	Extension of X bits in a constant.
W343	VERILINT	Extension of Z bits in a constant.
W359	VERILINT	For - condition expression is constant.
W372	VERILINT	Undefined PLI task.
W373	VERILINT	Undefined PLI function.
W389	VERILINT	<% value> clocks in the module.
W390	VERILINT	Multiple resets in the module.
W392	VERILINT	Wrong reset polarity.
W394	VERILINT	Multiple clocks in the always block.
W396	VERILINT	A flip-flop without reset.
W397	VERILINT	Destination bit is input.
W401	VERILINT	Clock <%item> is not an input to the module.
W402	VERILINT	Reset <%item> is not an input to the module.
W403	VERILINT	Clock is used as data.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
W410	VERILINT	Latch inferred for <%item>.
W414	VERILINT	Non blocking assignment in combinational block.
W415	VERILINT	Multiple drivers to net <%item> detected.
W416	VERILINT	Instance connection not by name.
W421	VERILINT	Non event-control statement (@) in always block.
W424	VERILINT	Functions sets a global variable <%item>.
W425	VERILINT	Functions uses a global variable <%item>.
W426	VERILINT	Tasks sets a global variable <%item>.
W427	VERILINT	Tasks uses a global variable <%item>.
W43	VERILINT	Wait statement may not be synthesizable.
W430	VERILINT	Initial statement may not be synthesizable.
W434	VERILINT	Top level module is a primitive.
W438	VERILINT	Tristate is not in a top level module.
W443	VERILINT	X in based number constant.
W444	VERILINT	High Z in based number constant.
W445	VERILINT	Output or inout <%item> tied to supply.
W446	VERILINT	Reading from an output port <%item>.
W450	VERILINT	Multi-bit expression (e.g a[2:0] used as clock.
W455	VERILINT	Not all cases are covered in full case.
W456	VERILINT	Variable <%item> is in the sensitivity list but not used in the block.
W459	VERILINT	Constant is extended to the implied width of 32 bits.
W467	VERILINT	'?' in based number constant.
W468	VERILINT	Index variable is too short.
W473	VERILINT	A port <%item> without range is re-declared with a range.
W478	VERILINT	Bad loop initialization statement.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
W479	VERILINT	Bad loop step statement.
W483	VERILINT	Assigned to self. This could imply a latch in synthesis.
W484	VERILINT	Possible loss of carry/borrow in addition/subtraction.
W485	VERILINT	Non-negative (reg) is compared to 0.
W488	VERILINT	Bus variable in the sensitivity list but not all its bits are used in the block.
W489	VERILINT	Last function statement does not assign to the function.
W490	VERILINT	Tristate control expression is not a variable name.
W491	VERILINT	Extension of ? bits in a constant.
W496	VERILINT	Comparison to 3 state are treated as false.
W499	VERILINT	Last function statement does not assign to all the bits of the function.
W502	VERILINT	A variable in the sensitivity list is modified inside the block.
W504	VERILINT	Integer <%item> is used in port expression.
W505	VERILINT	Mixed assignment styles (delay and non-blocking).
W507	VERILINT	Too many strengths for a pullup/pulldown gate (only one is needed).
W509	VERILINT	Defparam may not be synthesizable.
W521	VERILINT	Not all the bits of the variable are in the sensitivity list.
W526	VERILINT	Nested ifs. Consider using case or casex statement instead.
W527	VERILINT	'if' without an 'else' when one may be expected (dangling 'else' for a nested 'if'). Make sure the nesting is correct.
W529	VERILINT	'ifdef may not be supported by some synthesis tools.
W531	VERILINT	Truncating leading zeros (or x's or z's).
W541	VERILINT	Tristate is inferred.
W547	VERILINT	Redundant case expression.
W548	VERILINT	Synchronous flip-flop is inferred.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
W549	VERILINT	Asynchronous flip-flop is inferred.
W550	VERILINT	Mux is inferred.
W551	VERILINT	full_case has a default clause.
W554	VERILINT	Unconventional assigning to a function. Consider using regular assignment statement ('=').
W555	VERILINT	Unconventional deassigning to a function.
W556	VERILINT	Complex condition expression. Could be as a result of wrong interpretation of operator precedence.
W557	VERILINT	Illegal use of range for scalar parameter.
W558	VERILINT	Illegal use of bit select for scalar parameter.
W561	VERILINT	Based number with 0 width is extended to the implied width of 32 bits.
W562	VERILINT	Variable is assigned in both blocking and non-blocking assignments.
W563	VERILINT	Reduction of a single bit expression is redundant.
W565	VERILINT	Inferred a shift register.
W570	VERILINT	Inferred a counter.
W575	VERILINT	Logical NOT_OP operating on a vector.
W576	VERILINT	Multibit operand in a logical expression.
W592	VERILINT	Constant (parameter or specparam) is used in event control expression.
W594	VERILINT	Not all cases are covered in full case, but default case exists.
W599	VERILINT	This construct is not supported by Synopsys.
W601	VERILINT	The loop index is being modified.
W631	VERILINT	Assigned to self. This is harmless, but can reduce simulation speed.
W639	VERILINT	For synthesis, operands of a division or modulo operation need to be constants.
W67	VERILINT	Not a constant expression.

Table 39: Leda-classic Prebuilt Configuration (Continued)

Rule Label	Policy	Message
W69	VERILINT	Case statement without default clause but all the cases are covered.
W71	VERILINT	Case statement without default clause and not all cases are covered.

CDC Prebuilt Configuration

The following rules are from the CDC prebuilt configuration. CDC rules check clock domain crossings. This configuration contains about 32 rules drawn from the DESIGN policy. To load this rule configuration, from the Rule Wizard, choose **Config > Load configuration**, and use the pull-down menu to select CDC.

Table 40: CDC Prebuilt Configuration

Rule Label	Policy	Message
NTL_CLK05	DESIGN	Data must be registered by 2 or more flip-flops when crossing clock domain.
NTL_CLK23	DESIGN	Multiple asynchronous clock domain signals converging on <gate name>.
NTL_CLK24	DESIGN	Multibit control signal crossing clock domain should be Gray coded.
NTL_CLK25	DESIGN	Control signal crossing clock domain.
NTL_CLK26	DESIGN	Control signal crossing clock domain with data transfer.
NTL_CLK27	DESIGN	Control signal crossing clock domain without data transfer.
NTL_CLK29	DESIGN	Primary input feeds multiple clock domain.
NTL_CLK30	DESIGN	Reset is used in multiple clock domain.
NTL_CLK31	DESIGN	Clock signal is connected to the select pin of the MUX.
NTL_CLK33	DESIGN	Detect the combinational circuit other than selector (multiplexor) which merges the multiple clocks.
NTL_CLK34	DESIGN	Do not use meta-stable flip-flop
NTL_PAR13	DESIGN	Separate the design according to clock domains.
NTL_PAR17	DESIGN	Asynchronous parts should be placed in separate entities.
NTL_STR14	DESIGN	Check the circuits labeled _meta are really proper metastable circuits.
NTL_STR15	DESIGN	Give unique name to synchronizers so that they can be identified.
NTL_STR86	DESIGN	No Fanout within synchronizer.

SDC-postlayout Prebuilt Configuration

- The following rules are from the SDC-postlayout prebuilt configuration. This configuration contains about 36 rules drawn from the CONSTRAINTS policy. To load this rule configuration, from the Rule Wizard, choose **Config > Load configuration**, and use the pull-down menu to select SDC-postlayout. A duplicate of this configuration is also available as sdc-quality-postlayout prebuilt configuration. To load this rule configuration, from the Rule Wizard, choose **Config > Load configuration**, and use the pull-down menu to select SDC-quality-postlayout.

Table 41: SDC-postlayout Prebuilt Configuration

Rule Label	Policy	Message
SDC_CLK02	CONSTRAINTS	Unused clock constraint.
SDC_CLK03	CONSTRAINTS	Generated clock is not in the transitive fanout of its master clock.
SDC_CLK04	CONSTRAINTS	Generated clock master is not used as clock in design.
SDC_CLK08	CONSTRAINTS	Source pin of generated clock is not a port of design.
SDC_CLK09	CONSTRAINTS	Source pin of generated clock is in the fanout of the source pin of another clock, but is not generated by the later.
SDC_CLK10	CONSTRAINTS	Clock has no set_propagated_clock constraint in postlayout.
SDC_CLK14	CONSTRAINTS	Incomplete clock definition: both -waveform and -period are missing.
SDC_CLK15	CONSTRAINTS	Incomplete generated clock definition: one of -divide_by, -multiply_by, -invert, -edge_shift or edges must be present.
SDC_CLK20	CONSTRAINTS	Same clock has multiple definitions.
SDC_CTR06	CONSTRAINTS	Undefined clock transition for real clock.
SDC_CTR08	CONSTRAINTS	Do not use set_clock_transition in postlayout; use set_input_transiton instead.
SDC_CTR10	CONSTRAINTS	set_driving_cell on clock ports is not recommended in postlayout.
SDC_CTR12	CONSTRAINTS	Incomplete set_input_transition option.
SDC_FLP01_A	CONSTRAINTS	False path reference points are not connected.
SDC_FLP01_B	CONSTRAINTS	False path reference points do not exist.
SDC_IDL01	CONSTRAINTS	Unconstrained input.

Table 41: SDC-postlayout Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SDC_IDL04	CONSTRAINTS	Incomplete set_input_delay options.
SDC_IDL05	CONSTRAINTS	Inconsistent set_input_delay value versus clock period.
SDC_IDL06	CONSTRAINTS	Input constrained versus wrong (real) clock.
SDC_ITR01	CONSTRAINTS	Undefined set_input_transition or set_driving_cell.
SDC_ITR02	CONSTRAINTS	Incomplete input_transition options.
SDC_ITR03	CONSTRAINTS	Inconsistent set_input_transition option values -min > max.
SDC_ITR07	CONSTRAINTS	Unusual input transition options.
SDC_ITR09	CONSTRAINTS	Negative set_input_transition value.
SDC_MCP01_A	CONSTRAINTS	Multicycle path reference points are not connected.
SDC_MCP01_B	CONSTRAINTS	Multicycle path reference points do not exist.
SDC_NAM01	CONSTRAINTS	Do not name clocks same as port or pin name.
SDC_ODL01	CONSTRAINTS	Unconstrained output.
SDC_ODL04	CONSTRAINTS	Incomplete set_output_delay options.
SDC_ODL05	CONSTRAINTS	Inconsistent output delay versus clock period.
SDC_ODL06	CONSTRAINTS	Output constrained versus wrong (real) clock.
SDC_ODL12	CONSTRAINTS	Unusual output delay value.
SDC_OLD01	CONSTRAINTS	Undefined or zero load on output or inout port.
SDC_UNC01	CONSTRAINTS	Undefined clock uncertainty or zero clock uncertainty (real and generated clocks).
SDC_UNC02	CONSTRAINTS	Clock uncertainty is set on an object that is not a clock (real and generated clocks).
SDC_UNC05	CONSTRAINTS	Negative clock uncertainty value.

SDC-prelayout Prebuilt Configuration

The following rules are from the SDC-prelayout prebuilt configuration. This configuration contains about 46 rules drawn from the CONSTRAINTS policy. To load this rule configuration, from the Rule Wizard, choose **Config > Load configuration**, and use the pull-down menu to select SDC-prelayout. A duplicate of this configuration is also available as sdc-quality-prelayout prebuilt configuration. To load this rule configuration, from the Rule Wizard, choose **Config > Load configuration**, and use the pull-down menu to select SDC-quality-prelayout.

Table 42: SDC-prelayout Prebuilt Configuration

Rule Label	Policy	Message
SDC_CLK01	CONSTRAINTS	Unconstrained clock. No create_clock or create_generated_clock found.
SDC_CLK02	CONSTRAINTS	Unused clock constraint.
SDC_CLK03	CONSTRAINTS	Generated clock is not in the transitive fanout of its master clock.
SDC_CLK04	CONSTRAINTS	Generated clock master is not used as clock in design.
SDC_CLK08	CONSTRAINTS	Source pin of generated clock is not a port of design.
SDC_CLK09	CONSTRAINTS	Source pin of generated clock is in the fanout of the source pin of another clock, but is not generated by the later.
SDC_CLK11	CONSTRAINTS	set_propagated_clock defined on clock in prelayout.
SDC_CLK14	CONSTRAINTS	Incomplete clock definition: both -waveform and -period are missing.
SDC_CLK15	CONSTRAINTS	Incomplete generated clock definition: one of -divide_by, -multiply_by, -invert, -edge_shift or edges must be present.
SDC_CLK20	CONSTRAINTS	Same clock line has multiple definitions.
SDC_CTR01	CONSTRAINTS	Missing set_clock_transition constraint for clock.
SDC_CTR02	CONSTRAINTS	set_clock_transition is set on an object that is not a clock.
SDC_CTR09	CONSTRAINTS	set_input_transition and set_driving_cell on clock ports are not recommended in prelayout.
SDC_CTR11	CONSTRAINTS	Incomplete set_clock_transition options.
SDC_FLP01_A	CONSTRAINTS	False path reference points are not connected.
SDC_FLP01_B	CONSTRAINTS	False path reference points do not exist.

Table 42: SDC-prelayout Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SDC_IDL01	CONSTRAINTS	Unconstrained input.
SDC_IDL04	CONSTRAINTS	Incomplete set_input_delay options.
SDC_IDL05	CONSTRAINTS	Inconsistent set_input_delay value versus clock period.
SDC_IDL06	CONSTRAINTS	Input constrained versus wrong (real) clock.
SDC_ITR01	CONSTRAINTS	Undefined set_input_transition or set_driving_cell.
SDC_ITR02	CONSTRAINTS	Incomplete input_transition options.
SDC_ITR03	CONSTRAINTS	Inconsistent set_input_transition option values -min > max.
SDC_ITR07	CONSTRAINTS	Unusual input transition options.
SDC_ITR09	CONSTRAINTS	Negative set_input_transition value.
SDC_LAT01	CONSTRAINTS	Undefined clock latency or zero clock latency for real clocks.
SDC_LAT02	CONSTRAINTS	Clock latency is set on an object that is not a clock.
SDC_LAT03	CONSTRAINTS	Source latency for a generated clock is less than or equal to the source clock latency.
SDC_LAT06	CONSTRAINTS	Undefined source latency or zero source latency for generated clock.
SDC_LAT07_A	CONSTRAINTS	Incomplete set_clock_latency options.
SDC_LAT07_B	CONSTRAINTS	Incomplete set_clock_latency options.
SDC_LAT08_A	CONSTRAINTS	Inconsistent clock latency option values -min > max.
SDC_LAT08_B	CONSTRAINTS	Inconsistent clock latency option values -early > late.
SDC_LAT09	CONSTRAINTS	Negative clock latency value.
SDC_MCP01_A	CONSTRAINTS	Multicycle path reference points are not connected.
SDC_MCP01_B	CONSTRAINTS	Multicycle path reference points do not exist.
SDC_NAM01	CONSTRAINTS	Do not name clocks same as port or pin name.
SDC_ODL01	CONSTRAINTS	Unconstrained output.
SDC_ODL04	CONSTRAINTS	Incomplete set_output_delay options.

Table 42: SDC-prelayout Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SDC_ODL05	CONSTRAINTS	Inconsistent output delay versus clock period.
SDC_ODL06	CONSTRAINTS	Output constrained versus wrong (real) clock.
SDC_ODL12	CONSTRAINTS	Unusual output delay value.
SDC_OLD01	CONSTRAINTS	Undefined or zero load on output or inout port.
SDC_UNC01	CONSTRAINTS	Undefined clock uncertainty or zero clock uncertainty (real and generated clocks).
SDC_UNC02	CONSTRAINTS	Clock uncertainty is set on an object that is not a clock (real and generated clocks).
SDC_UNC05	CONSTRAINTS	Negative clock uncertainty value.

SDC-RTL Prebuilt Configuration

The following rules are from the SDC-RTL prebuilt configuration. This configuration contains about 46 rules drawn from the CONSTRAINTS policy. To load this rule configuration, from the Rule Wizard, choose **Config > Load configuration**, and use the pull-down menu to select SDC-RTL. A duplicate of this configuration is also available as sdc-quality-rtl prebuilt configuration. To load this rule configuration, from the Rule Wizard, choose **Config > Load configuration**, and use the pull-down menu to select sdc-quality-rtl

Table 43: SDC-RTL Prebuilt Configuration

Rule Label	Policy	Message
SDC_CLK01	CONSTRAINTS	Unconstrained clock. No create_clock or create_generated_clock found.
SDC_CLK02	CONSTRAINTS	Unused clock constraint.
SDC_CLK03	CONSTRAINTS	Generated clock is not in the transitive fanout of its master clock.
SDC_CLK04	CONSTRAINTS	Generated clock master is not used as clock in design.
SDC_CLK08	CONSTRAINTS	Source pin of generated clock is not a port of design.
SDC_CLK09	CONSTRAINTS	Source pin of generated clock is in the fanout of the source pin of another clock, but is not generated by the later.
SDC_CLK11	CONSTRAINTS	set_propagated_clock defined on clock in prelayout.
SDC_CLK14	CONSTRAINTS	Incomplete clock definition: both -waveform and -period are missing.
SDC_CLK15	CONSTRAINTS	Incomplete generated clock definition: one of -divide_by, -multiply_by, -invert, -edge_shift or edges must be present.
SDC_CLK20	CONSTRAINTS	Same clock line has multiple definitions.
SDC_CTR01	CONSTRAINTS	Missing set_clock_transition constraint for clock.
SDC_CTR02	CONSTRAINTS	set_clock_transition is set on an object that is not a clock.
SDC_CTR09	CONSTRAINTS	set_input_transition and set_driving_cell on clock ports are not recommended in prelayout.
SDC_CTR11	CONSTRAINTS	Incomplete set_clock_transition options.
SDC_FLP01_A	CONSTRAINTS	False path reference points are not connected.
SDC_FLP01_B	CONSTRAINTS	False path reference points do not exist.

Table 43: SDC-RTL Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SDC_IDL01	CONSTRAINTS	Unconstrained input.
SDC_IDL04	CONSTRAINTS	Incomplete set_input_delay options.
SDC_IDL05	CONSTRAINTS	Inconsistent set_input_delay value versus clock period.
SDC_IDL06	CONSTRAINTS	Input constrained versus wrong (real) clock.
SDC_ITR01	CONSTRAINTS	Undefined set_input_transition or set_driving_cell.
SDC_ITR02	CONSTRAINTS	Incomplete input_transition options.
SDC_ITR03	CONSTRAINTS	Inconsistent set_input_transition option values -min > max.
SDC_ITR07	CONSTRAINTS	Unusual input transition options.
SDC_ITR09	CONSTRAINTS	Negative set_input_transition value.
SDC_LAT01	CONSTRAINTS	Undefined clock latency or zero clock latency for real clocks.
SDC_LAT02	CONSTRAINTS	Clock latency is set on an object that is not a clock.
SDC_LAT03	CONSTRAINTS	Source latency for a generated clock is less than or equal to the source clock latency.
SDC_LAT06	CONSTRAINTS	Undefined source latency or zero source latency for generated clock.
SDC_LAT07_A	CONSTRAINTS	Incomplete set_clock_latency options.
SDC_LAT07_B	CONSTRAINTS	Incomplete set_clock_latency options.
SDC_LAT08_A	CONSTRAINTS	Inconsistent clock latency option values -min > max.
SDC_LAT08_B	CONSTRAINTS	Inconsistent clock latency option values -early > late.
SDC_LAT09	CONSTRAINTS	Negative clock latency value.
SDC_MCP01_A	CONSTRAINTS	Multicycle path reference points are not connected.
SDC_MCP01_B	CONSTRAINTS	Multicycle path reference points do not exist.
SDC_NAM01	CONSTRAINTS	Do not name clocks same as port or pin name.
SDC_ODL01	CONSTRAINTS	Unconstrained output.
SDC_ODL04	CONSTRAINTS	Incomplete set_output_delay options.

Table 43: SDC-RTL Prebuilt Configuration (Continued)

Rule Label	Policy	Message
SDC_ODL05	CONSTRAINTS	Inconsistent output delay versus clock period.
SDC_ODL06	CONSTRAINTS	Output constrained versus wrong (real) clock.
SDC_ODL12	CONSTRAINTS	Unusual output delay value.
SDC_OLD01	CONSTRAINTS	Undefined or zero load on output or inout port.
SDC_UNC01	CONSTRAINTS	Undefined clock uncertainty or zero clock uncertainty (real and generated clocks).
SDC_UNC02	CONSTRAINTS	Clock uncertainty is set on an object that is not a clock (real and generated clocks).
SDC_UNC05	CONSTRAINTS	Negative clock uncertainty value.

SDC-top-versus-block Prebuilt Configuration

The following rules are from the SDC-top-versus-block prebuilt configuration. This configuration contains about five rules drawn from the CONSTRAINTS policy. To load this rule configuration, from the Rule Wizard, choose **Config > Load configuration**, and use the pull-down menu to select SDC-top-versus-block.

Table 44: SDC-top-versus-block Prebuilt Configuration

Rule Label	Policy	Message
SDC_TOP01	CONSTRAINTS	Block level clock constraint is inconsistent with top level clock constraint.
SDC_TOP02	CONSTRAINTS	Block level I/O delay constraint is inconsistent with top level I/O delay constraint.
SDC_TOP03	CONSTRAINTS	Block level false path constraint is inconsistent with top level false path constraint.
SDC_TOP04	CONSTRAINTS	Block level multicycle path constraint is inconsistent with top level multicycle path constraint.
SDC_TOP20	CONSTRAINTS	Block level max/min delay constraint is inconsistent with top level max/min delay constraint.

SDC-equivalency Prebuilt Configuration

The following rules are from the SDC-equivalency prebuilt configuration. This configuration contains about 14 rules drawn from the CONSTRAINTS policy. To load this rule configuration, from the Rule Wizard, choose **Config > Load configuration**, and use the pull-down menu to select SDC-equivalency.

Table 45: SDC-equivalency Prebuilt Configuration

Rule Label	Policy	Message
SDC_EQCLK01	CONSTRAINTS	Equivalency file clock constraint is inconsistent with reference file clock constraint: equivalency clock is %s
SDC_EQCLK02	CONSTRAINTS	Reference file clock constraint is inconsistent with equivalency file clock constraint: reference clock is %s
SDC_EQIDL01	CONSTRAINTS	Equivalency file input delay constraint is inconsistent with reference file delay constraint: input port is %s
SDC_EQIDL02	CONSTRAINTS	Reference file input delay constraint is inconsistent with equivalency file delay constraint: input port is %s
SDC_EQODL01	CONSTRAINTS	Equivalency file output delay constraint is inconsistent with reference file delay constraint: output port is %s
SDC_EQODL02	CONSTRAINTS	Reference file output delay constraint is inconsistent with equivalency file delay constraint: output port is %s
SDC_EQFLP01	CONSTRAINTS	Equivalency file false path constraint is inconsistent with reference file false path constraint.
SDC_EQFLP02	CONSTRAINTS	Reference file false path constraint is inconsistent with equivalency file false path constraint.
SDC_EQMCP01	CONSTRAINTS	Equivalency file multicycle path constraint is inconsistent with reference file multicycle path constraint.
SDC_EQMCP02	CONSTRAINTS	Reference file multicycle path constraint is inconsistent with equivalency file multicycle path constraint.
SDC_EQCMB01	CONSTRAINTS	Equivalency file max-delay path constraint is inconsistent with reference file max-delay path constraint.
SDC_EQCMB02	CONSTRAINTS	Equivalency file min-delay path constraint is inconsistent with reference file min-delay path constraint.
SDC_EQCMB03	CONSTRAINTS	Reference file max-delay path constraint is inconsistent with equivalency file max-delay path constraint.
SDC_EQCMB04	CONSTRAINTS	Reference file min-delay path constraint is inconsistent with equivalency file min-delay path constraint.

D

Leda Duplicated Rules

Introduction

Leda contains a list of 226 rules that appears in several different policies. These rules are called Redundant/Duplicated rules. You can disable the redundant rules from the Rule Wizard to avoid getting duplicated error messages.

Disabling Redundant Rules

The Topics tab on the left side of Rule Wizard gives you an easy way to review all of the rules in the different policies related to a given topic. For example, if you click on the (+) icon next to the Clocks topic on the left side of the display, the tree expands to show a list of rules associated with clocks. One such rule is to avoid the use of both positive- and negative-edge triggered flip-flops in the same design. Because this is a good common sense design, rule, it appears in several different policies. When you click on the (+) icon just to the left of the description for this rule, the display expands to show the different policies where this rule is available, including DFT, RMM, and STARC.

Let's say that you want a rule to be enabled for checking, but you don't want five different error messages to appear just because the rule is duplicated in five different policies. To narrow your error report display, you can use the Topics tab to view redundant rules. Click on the rule you want to disable and deselect it for checking using the check box on the right hand side of the Rule Wizard. Do this for all but one of the redundant rules, leaving just one relevant rule enabled for checking. To disable all redundant rules at once, first run the Checker, right click on the rule in the Error Viewer, and select "Disable Redundant Rules" from the pop-up menu.

Duplicated Rule List

The duplicated rules are listed with the description and a table containing information about the policy of the respective rules.



Note

These information of the Duplicated rules include for both languages VHDL and Verilog. Please check for the selected language, when you select one of them from this duplicated rule list.

For example, rules B_3608, VER_2_11_1_4, and VHD_2_11_1_4 all check for the same violation “The number of states in a state machine should be less than 40”. Here rule B_3608 supports both VHDL and Verilog but, VER_2_11_1_4 supports only Verilog, and VHD_2_11_1_4 supports only VHDL.

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Port default values are ignored	SYN1_1_2	IEEE_RTL_SYNTH_SUBSET
	VHD_2_1_3_4	VHD_STARC_DSG
Only generics of type Integer for synthesis	DCVHDL_2024	DC
	SYN1_1_3	IEEE_RTL_SYNTH_SUBSET
	VHD_3_2_4_1	VHD_STARC_DSG
Real data types is not synthesizable.	DCVHDL_2108	DC
	SYN2_2_4	IEEE_RTL_SYNTH_SUBSET
Signal declaration in a package (global signal) is not supported in cycle mode.	SC_027	SCIROCCO_CYCLE
	SYN2_5_2_F	IEEE_RTL_SYNTH_SUBSET
File declarations are illegal in package declarations.	SC_028	SCIROCCO_CYCLE
	SYN2_5_2_A	IEEE_RTL_SYNTH_SUBSET

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Physical types are not supported for synthesis.	DCVHDL_2095	DC
	SYN3_1_2	IEEE_RTL_SYNTH_SUBSET
Multi-dimensional arrays are not supported for synthesis.	DCVHDL_2140	DC
	SYN3_2_1_A	IEEE_RTL_SYNTH_SUBSET
Access types are not supported for synthesis.	DCVHDL_2093	DC
	SYN3_3_1	IEEE_RTL_SYNTH_SUBSET
File type definitions are illegal	VHD_2_1_10_9	VHD_STARC_DSG
	DCVHDL_2094	DC
	SYN3_4_1	IEEE_RTL_SYNTH_SUBSET
Incomplete type declarations are ignored.	DCVHDL_2096	DC
	SC_009	SCIROCCO_CYCLE
	SYN4_1_1_A	IEEE_RTL_SYNTH_SUBSET
Deferred constants are not supported for synthesis.	DCVHDL_2155	DC
	SC_006	SCIROCCO_CYCLE
	SYN4_3_1_1_1	IEEE_RTL_SYNTH_SUBSET
Initial values for signals are not supported for synthesis.	DCVHDL_2022	DC
	SYN4_3_1_2_1	IEEE_RTL_SYNTH_SUBSET
Bus signal kind is ignored.	SC_100	SCIROCCO_CYCLE
	SYN4_3_1_2_2_A	IEEE_RTL_SYNTH_SUBSET
Register signal kind is ignored.	SC_101	SCIROCCO_CYCLE
	SYN4_3_1_2_2_B	IEEE_RTL_SYNTH_SUBSET

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Initial values for variable declarations are ignored.	DCVHDL_228	DC
	SYN4_3_1_3_1	IEEE_RTL_SYNTH_SUBSET
Shared variable declarations are illegal.	SC_010	SCIROCCO_CYCLE
	SYN4_3_1_3_2	IEEE_RTL_SYNTH_SUBSET
	VHD_2_1_10_3	VHD_STARC_DSG
File declarations are not supported for synthesis.	DCVHDL_2042	DC
	SYN4_3_1_4_1	IEEE_RTL_SYNTH_SUBSET
Alias declarations are ignored.	DCVHDL_2041	DC
	SYN4_3_3_1	IEEE_RTL_SYNTH_SUBSET
	VHD_2_1_10_10	VHD_STARC_DSG
Configuration specifications are not supported for synthesis.	DCVHDL_2091	DC
	SYN5_2_1	IEEE_RTL_SYNTH_SUBSET
Disconnect specification is ignored.	DCVHDL_2043	DC
	SC_104	SCIROCCO_CYCLE
	SYN5_3_1	IEEE_RTL_SYNTH_SUBSET
	VHD_2_1_10_12	VHD_STARC_DSG
Standard shift operations not allowed.	SYN7_2_2	IEEE_RTL_SYNTH_SUBSET
	VHD_2_1_4_3	VHD_STARC_DSG
Record aggregates are illegal.	DCVHDL_2111	DC
	SYN7_3_2_1_1	IEEE_RTL_SYNTH_SUBSET

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Timeout clauses are ignored in wait statements.	DCVHDL_2050	DC
	SYN8_1_4	IEEE_RTL_SYNTH_SUBSET
Multiple waveform elements are not supported.	SC_016	SCIROCCO_CYCLE
	SYN8_4_1	IEEE_RTL_SYNTH_SUBSET
Delays in signal assignment are ignored by synthesis tool.	B_2008	LEDA
	SYN8_4_1_2	IEEE_RTL_SYNTH_SUBSET
Keyword “reject” is not supported.	SC_019	SCIROCCO_CYCLE
	SYN8_4_4	IEEE_RTL_SYNTH_SUBSET
while loops are not supported.	DCVHDL_165	DC
	SYN8_9_3	IEEE_RTL_SYNTH_SUBSET
	VHD_2_1_10_4	VHD_STARC_DSG
Guard expressions not allowed in block statements.	DCVHDL_2045	DC
	SC_032	SCIROCCO_CYCLE
	SYN9_1_1	IEEE_RTL_SYNTH_SUBSET
Variables must be initialized before being used (to prevent latch inference).	B_2011	LEDA
	DCHDL_177	DC
	FM_2_3	FORMALITY
	SC_202	SCIROCCO_CYCLE
	SYN9_2_3	IEEE_RTL_SYNTH_SUBSET
Entity names are not supported in component instantiation statements.	SYN9_6_1	IEEE_RTL_SYNTH_SUBSET
	VHD_3_2_3_3	VHD_STARC_DSG

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Configuration names are not supported in component instantiation statements.	DCVHDL_2131	DC
	SYN9_6_2	IEEE_RTL_SYNTH_SUBSET
Block declarative part in generate statement is not supported.	DCVHDL_2284	DC
	SYN9_7_1	IEEE_RTL_SYNTH_SUBSET
Do not assign to a global variable in a function.	CS_7C_R_B	DESIGNWARE
	SYN10_1	IEEE_RTL_SYNTH_SUBSET
	VER_2_1_3_5	VER_STARC_DSG
	W424	VERILINT
Do not assign to a global variable in a task.	B_3402	LEDA
	CS_8C_R	DESIGNWARE
	SYN10_2	IEEE_RTL_SYNTH_SUBSET
	W426	VERILINT
Incompatible port connection in module instantiation.	DCVER_4	DC
	SYN12_2	IEEE_RTL_SYNTH_SUBSET
Specify blocks are ignored.	DCVER_276	DC
	SYN13_1	IEEE_RTL_SYNTH_SUBSET
System task enables are not allowed.	B_2000	LEDA
	SYN14_1	IEEE_RTL_SYNTH_SUBSET
TRIREG declarations are not supported by synthesis.	DCVER_183	DC
	SYN3_2_1_B	IEEE_RTL_SYNTH_SUBSET
	VCS_2_4	VCS

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
TRIOR declarations are not supported by synthesis.	DCVER_180	DC
	SYN3_2_10	IEEE_RTL_SYNTH_SUBSET
Drive strengths in net declaration are ignored.	DCVER_977	DC
	E304	VERILINT
	SYN3_2_2	IEEE_RTL_SYNTH_SUBSET
Charge strengths are ignored.	DCVER_277	DC
	SYN3_2_3	IEEE_RTL_SYNTH_SUBSET
TRIREG declarations are not supported for synthesis	DCVER_182	DC
	SYN3_2_7	IEEE_RTL_SYNTH_SUBSET
TRIAND declarations are not supported by synthesis	DCVER_179	DC
	SYN3_2_8	IEEE_RTL_SYNTH_SUBSET
TRIO declarations are not supported by synthesis.	DCVER_181	DC
	SYN3_2_9	IEEE_RTL_SYNTH_SUBSET
Time declarations are not supported for synthesis.	DCVER_191	DC
	SYN3_9_1	IEEE_RTL_SYNTH_SUBSET
	VCS_2_2	VCS
Real declarations are not supported for synthesis.	DCVER_177	DC
	SYN3_9_2	IEEE_RTL_SYNTH_SUBSET
	W294	VERILINT
Realtime declarations are not supported for synthesis.	DCVER_178	DC
	SYN3_9_3	IEEE_RTL_SYNTH_SUBSET

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
CASE EQUALITY (===) is not supported by synthesis.	DCVER_189	DC
	SYN4_1_2	IEEE_RTL_SYNTH_SUBSET
	FM_2_25	FORMALITY
CASE INEQUALITY (===) is not supported by synthesis.	DCVER_190	DC
	SYN4_1_3	IEEE_RTL_SYNTH_SUBSET
	FM_2_26	FORMALITY
Do not use assignment in net/signal declaration.	FM_2_6B	FORMALITY
	SYN6_1_1	IEEE_RTL_SYNTH_SUBSET
Drive strength specification for continuous assignment is ignored.	DCVER_309	DC
	SYN6_1_2	IEEE_RTL_SYNTH_SUBSET
	VCS_4	VCS
Delays for continuous assignment are ignored.	DCVER_173	DC
	SYN6_1_5	IEEE_RTL_SYNTH_SUBSET
NMOS switches are not supported.	DCVER_296	DC
	SYN7_1_1	IEEE_RTL_SYNTH_SUBSET
RTRANIF1 switches are not supported.	DCVER_270	DC
	SYN7_1_10	IEEE_RTL_SYNTH_SUBSET
CMOS switches are not supported.	DCVER_295	DC
	SYN7_1_11	IEEE_RTL_SYNTH_SUBSET
RCMOS switches are not supported.	DCVER_265	DC
	SYN7_1_12	IEEE_RTL_SYNTH_SUBSET

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
pull (pullup and pulldown) gate instantiations are not supported for synthesis.	VCS_3_7	VCS
	SYN7_1_13	IEEE_RTL_SYNTH_SUBSET
Drive strength specification for tristate gate instantiation is ignored.	DCVER_306	DC
	SYN7_1_16	IEEE_RTL_SYNTH_SUBSET
PMOS switches are not supported.	DCVER_297	DC
	SYN7_1_2	IEEE_RTL_SYNTH_SUBSET
RNMOS switches are not supported.	DCVER_266	DC
	SYN7_1_3	IEEE_RTL_SYNTH_SUBSET
RPMOS switches are not supported.	DCVER_267	DC
	SYN7_1_4	IEEE_RTL_SYNTH_SUBSET
TRAN switches are not supported.	DCVER_271	DC
	SYN7_1_5	IEEE_RTL_SYNTH_SUBSET
RTRAN switches are not supported.	DCVER_268	DC
	SYN7_1_6	IEEE_RTL_SYNTH_SUBSET
TRANIF0 switches are not supported.	DCVER_272	DC
	SYN7_1_7	IEEE_RTL_SYNTH_SUBSET
TRANIF1 switches are not supported.	DCVER_273	DC
	SYN7_1_8	IEEE_RTL_SYNTH_SUBSET
RTRANIF0 switches are not supported.	DCVER_269	DC
	SYN7_1_9	IEEE_RTL_SYNTH_SUBSET

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
User-defined primitives (UDPs) are not supported.	DCVER_275	DC
	SYN8_1_1_B	IEEE_RTL_SYNTH_SUBSET
Initial statement not supported.	DCVER_192	DC
	FM_2_6A	FORMALITY
Initial constructs are ignored by synthesis tools.	SYN9_16	IEEE_RTL_SYNTH_SUBSET
	VER_2_3_4_2	VER_STARC_DSG
	W430	VERILINT
Procedural-continuous assignments are not supported by synthesis.	DCVER_966	DC
	SYN9_17	IEEE_RTL_SYNTH_SUBSET
	W20	VERILINT
The 'deassign' construct is not supported by synthesis.	DCVER_969	DC
	SYN9_18	IEEE_RTL_SYNTH_SUBSET
	W21	VERILINT
The 'force' construct is not supported by synthesis.	DCVER_967	DC
	SYN9_19	IEEE_RTL_SYNTH_SUBSET
Missing or redundant signals in the process sensitivity list.	SC_107	SCIROCCO_CYCLE
	SYN9_2	IEEE_RTL_SYNTH_SUBSET
The 'release' construct is not supported by synthesis.	DCVER_968	DC
	SYN9_20	IEEE_RTL_SYNTH_SUBSET
Delay statements are ignored for synthesis.	DCVER_176	DC
	SYN9_22	IEEE_RTL_SYNTH_SUBSET

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Repeat constructs are not supported in synthesis.	DCVER_219	DC
	SYN9_24	IEEE_RTL_SYNTH_SUBSET
WAIT statements are not supported by synthesis.	DCVER_188	DC
	SYN9_28	IEEE_RTL_SYNTH_SUBSET
	W43	VERILINT
Event triggers not supported.	DCVER_193	DC
	SYN9_29	IEEE_RTL_SYNTH_SUBSET
	VCS_2_3	VCS
Do not mix blocking and non-blocking assignments in a combinational always block.	SYN9_3	IEEE_RTL_SYNTH_SUBSET
	VER_2_2_3_1	VER_STARC_DSG
FORK and JOIN constructs are not supported by synthesis.	DCVER_187	DC
	SYN9_30	IEEE_RTL_SYNTH_SUBSET
	VER_2_7_4_3	VER_STARC_DSG
Event declarations are not allowed.	B_2006	LEDA
	DCVER_286	DC
	SYN9_31	IEEE_RTL_SYNTH_SUBSET
Clk name should be clk or prefixed with clk.	B_4404	LEDA
	G_521_6	RMM_RTL_CODING_GUIDELINES
	N_2C_R_A	DESIGNWARE
	VER_1_1_5_2A	VER_STARC_DSG
	VHD_1_1_5_2A	VHD_STARC_DSG

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Process label should end with _PROC.	B_4223	LEDA
	G_5215_2	RMM_RTL_CODING_GUIDELINES
Begin instance name with U_.	B_4219	LEDA
	G_5215_4	RMM_RTL_CODING_GUIDELINES
	I_3C_G	DESIGNWARE
	VER_1_1_1_8	VER_STARC_DSG
	VHD_1_1_1_8	VHD_STARC_DSG
Types bit and bit_vector should not be used.	G_531_4	RMM_RTL_CODING_GUIDELINES
	VHD_2_1_2_4	VHD_STARC_DSG
Block statements are not allowed.	G_537_2	RMM_RTL_CODING_GUIDELINES
	VHD_2_1_10_1	VHD_STARC_DSG
Avoid using both positive-edge and negative-edge triggered flip-flops in your design.	DFT_003	DFT
	G_541_1	RMM_RTL_CODING_GUIDELINES
Use of both rising edge and falling edge triggered logic will yield extra cycle mode triggers which in turn will degrade simulation performance.	SC_204	SCIROCCO_CYCLE
	VER_1_2_1_1B	VER_STARC_DSG
	VHD_1_2_1_1B	VHD_STARC_DSG
Use signals instead of variables (suitable for synthesis).	G_556_1	RMM_RTL_CODING_GUIDELINES
	VHD_2_3_2_2	VHD_STARC_DSG

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Drivers of output port should be registered: %s	C_1005	LEDA
	DFT_009	DFT
	G_561_1	RMM_RTL_CODING_GUIDELINES
	S_6C_R_A	DESIGNWARE
	VER_1_6_2_1A	VER_STARC_DSG
	VHD_1_6_2_1A	VHD_STARC_DSG
Always use descending range for multi-bit signals and ports.	A_12C_R	DESIGNWARE
	R_521_10	RMM_RTL_CODING_GUIDELINES
	VER_2_1_6_1	VER_STARC_DSG
	VHD_2_1_6_1	VHD_STARC_DSG
Port order should be the following: in, out and inout.	R_5210_1_A	RMM_RTL_CODING_GUIDELINES
	R_5210_1_B	RMM_RTL_CODING_GUIDELINES
	VER_3_1_3_2	VER_STARC_DSG
	VHD_3_1_3_2	VHD_STARC_DSG
Instantiate module by name. Instantiating by position may cause errors.	FM_2_7	FORMALITY
	I_1C_R	DESIGNWARE
	R_5211_1	RMM_RTL_CODING_GUIDELINES
	VER_3_2_3_1	VER_STARC_DSG
	VHD_3_2_3_1	VHD_STARC_DSG
	W416	VERILINT
Process block should have a label.	B_4001	LEDA
	R_5215_1	RMM_RTL_CODING_GUIDELINES

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Module declarations should have header comments.	R_524_1_A	RMM_RTL_CODING_GUIDELINES
	VER_3_5_3_1	VER_STARC_DSG
	VHD_3_5_3_1	VHD_STARC_DSG
Do not use Verilog or VHDL keywords	R_529_1	RMM_RTL_CODING_GUIDELINES
	TK_6CP_R	DESIGNWARE
	VER_1_1_1_3A	VER_STARC_DSG
	VHD_1_1_1_3A	VHD_STARC_DSG
Redundant signal <%item> in sensitivity list.	B_3418	LEDA
	FM_2_1A	FORMALITY
	R_554_1_A	RMM_RTL_CODING_GUIDELINES
	W456	VERILINT
Do not use built-in Verilog primitive <%item>.	G_536_2	RMM_RTL_CODING_GUIDELINES
	I_4C_R	DESIGNWARE
Buffers must not be explicitly added to the clock path.	A_5C_R_A	DESIGNWARE
	C_1206	LEDA
	G_542_1	RMM_RTL_CODING_GUIDELINES
	VER_1_4_2_1	VER_STARC_DSG
	VHD_1_4_2_1	VHD_STARC_DSG
Always construct must start with event control (always @clk..).	G_551_1_B	RMM_RTL_CODING_GUIDELINES
	SYN9_32	IEEE_RTL_SYNTH_SUBSET
	VER_2_3_3_2	VER_STARC_DSG

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Only one asynchronous reset/set/load allowed in a sequential block.	G_551_1_I	RMM_RTL_CODING_GUIDELINES
	S_5C_R_B	DESIGNWARE
	C_1000	LEDA
	C_8C_R	DESIGNWARE
Avoid asynchronous feedback loop.	FM_1_1	FORMALITY
	G_553_1	RMM_RTL_CODING_GUIDELINES
	SC_305	SCIROCCO_CYCLE
	TEST_960	DFT
	VCS_1	VCS
	VER_1_2_1_3	VER_STARC_DSG
	VHD_1_2_1_3	VHD_STARC_DSG
A latch may be inferred for <%item>	DFT_021	DFT
	R_552_1	RMM_RTL_CODING_GUIDELINES
	SC_108	SCIROCCO_CYCLE
	SYN9_5	IEEE_RTL_SYNTH_SUBSET
	S_4C_R	DESIGNWARE
	VER_2_4_1_2	VER_STARC_DSG
	VHD_2_4_1_2	VHD_STARC_DSG
	W410	VERILINT

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
No blocking assignment are allowed in a sequential block.	B_3417	LEDA
	CS_5P_R_A	DESIGNWARE
	FM_2_15	FORMALITY
	R_555_1_A	RMM_RTL_CODING_GUIDELINES
	SYN9_7	IEEE_RTL_SYNTH_SUBSET
	VCS_14	VCS
	VER_2_3_1_1	VER_STARC_DSG
	W336	VERILINT
Use blocking assignments in combinatorial block.	B_3416	LEDA
	CS_5P_R_B	DESIGNWARE
	FM_2_16	FORMALITY
	R_555_1_B	RMM_RTL_CODING_GUIDELINES
	VCS_15	VCS
	W414	VERILINT
Reading from an output port <%item>.	B_1001	LEDA
	W446	VERILINT
Multiple non-tristate drivers to signal <%item> detected.	C_1009	LEDA
	FM_2_8	FORMALITY
	VCS_17	VCS
	VER_2_5_1_5	VER_STARC_DSG
	VHD_2_5_1_5	VHD_STARC_DSG
	W415	VERILINT

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Data must be registered by flipflops when changing clock domain.	B_1202	LEDA
	DFT_006	DFT
	VER_1_4_4_1	VER_STARC_DSG
	VHD_1_4_4_1	VHD_STARC_DSG
	W389	VERILINT
Do not use negative edge flipflop.	B_1203	LEDA
	VER_1_4_3_6	VER_STARC_DSG
	VHD_1_4_3_6	VHD_STARC_DSG
The clock signal <%item> is not coming directly from a port of the current unit.	B_1205	LEDA
	W401	VERILINT
Asynchronous reset/set/load <%item> exists in module/unit.	B_1404	LEDA
	DFT_019	DFT
	W549	VERILINT
Do not use active high asynchronous reset/set/load.	B_1407	LEDA
	VER_2_3_6_2	VER_STARC_DSG
	VHD_2_3_6_2	VHD_STARC_DSG
Do not use asynchronous reset and asynchronous set in same always block/process statement.	B_1409	LEDA
	VER_1_3_1_7	VER_STARC_DSG
	VHD_1_3_1_7	VHD_STARC_DSG
Shift by a non constant value is not allowed.	B_2001	LEDA
	W244	VERILINT
Non synthesizable operator === !== encountered.	B_2010	LEDA
	W339	VERILINT
Array of integer is not allowed.	B_3001	LEDA
	VCS_2_5	VCS

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Unequal length operand in comparison operator.	B_3201	LEDA
	VER_2_10_3_1	VER_STARC_DSG
	VHD_2_10_3_1	VER_STARC_DSG
Delay is not a constant expression.	B_3202	LEDA
	W129	VERILINT
? in based number constant is not allowed.	B_3204	LEDA
	W467	VERILINT
X in based number constant.	B_3206	LEDA
	W443	VERILINT
Empty block found. No statements in block.	B_3400	LEDA
	W192	VERILINT
Blocking delay not allowed in non-blocking assignment.	B_3401	LEDA
	DCVER_130	DC
Case statement should have a default case.	B_3403	LEDA
	VHD_2_8_1_4	VHD_STARC_DSG
while condition expression is constant.	B_3409	LEDA
	W228	VERILINT
Process must have a header comment.	B_4000	LEDA
	R_525_1_D	RMM_RTL_CODING_GUIDELINES
Subprogram must have a header comment.	B_4002	LEDA
	R_525_1_E	RMM_RTL_CODING_GUIDELINES
File names should be as follows <entity>.vhd.	B_4201	LEDA
	VHD_1_1_1_1	VHD_STARC_DSG

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Architecture names should be RTL, BEH, SIM(TB).	B_4202	LEDA
	VHD_1_1_6_1	VHD_STARC_DSG
Naming conventions for module name. Name should end in _MOD.	B_4203	LEDA
	MF_1C_R_A	DESIGNWARE.
	MF_1C_R_B	DESIGNWARE.
	MF_1C_R_C	DESIGNWARE.
	MF_1C_R_D	DESIGNWARE.
	MF_4C_R	DESIGNWARE
Entity and architecture should be in the same file.	B_4204	LEDA
	VHD_1_1_6_4	VHD_STARC_DSG
File names should be as follows: <module name>.v	B_4205	LEDA
	MF_3C_R	DESIGNWARE
	VER_1_1_1_1	VER_STARC_DSG
Naming conventions for package declaration file name. File name should be <package>.vhd	B_4207	LEDA
	VHD_1_1_4_1	VHD_STARC_DSG
Naming conventions for signal name. name should begin with S.	B_4211	LEDA
	VHD_1_1_3_1	VHD_STARC_DSG
Name all always blocks <name>_PROC.	B_4222	LEDA
	N_7C_G	DESIGNWARE

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Register name should end with _reg or _REG.	B_4403	LEDA
	G_521_13_B	RMM_RTL_CODING_GUIDELINES
	VER_1_1_5_1A	VER_STARC_DSG
	VER_1_1_5_1B	VER_STARC_DSG
	VHD_1_1_5_1A	VHD_STARC_DSG
	VHD_1_1_5_1B	VHD_STARC_DSG
Naming conventions for asynchronous reset. Name should begin with rst.	B_4405	LEDA
	N_2C_R_B	DESIGNWARE
Naming conventions for tristate signals. Name should end in _z.	B_4407	LEDA
	G_521_13_A	RMM_RTL_CODING_GUIDELINES
Only one clock allowed in the design. <% value> clock have been identified.	C_1200	LEDA
	SC_303	SCIROCCO_CYCLE
	VER_1_2_1_1A	VER_STARC_DSG
	VHD_1_2_1_1A	VHD_STARC_DSG
Clock is being used as data.	C_1201	LEDA
	DCHDL_175	DC
	VER_1_4_3_4	VER_STARC_DSG
	VHD_1_4_3_4	VHD_STARC_DSG
	W403	VERILINT
Data must be registered by 2 flipflops when changing clock domains.	A_3C_R	DESIGNWARE
	C_1202	LEDA
	VER_1_5_1_1	VER_STARC_DSG
	VHD_1_5_1_1	VHD_STARC_DSG

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Internally generated clock detected	C_1203	LEDA
	DFT_002	DFT
	G_544_1	RMM_RTL_CODING_GUIDELINES
	SC_301	SCIROCCO_CYCLE
	VER_1_4_3_2	VER_STARC_DSG
	VHD_1_4_3_2	VHD_STARC_DSG
No gated clock except in clock generator CKGEN.	C_1204	LEDA
	VER_1_4_1_1	VER_STARC_DSG
	VHD_1_4_1_1	VHD_STARC_DSG
Gated clocks are not allowed in the design.	A_5C_R_B	DESIGNWARE
	C_1207	LEDA
	G_543_1	RMM_RTL_CODING_GUIDELINES
	SC_300	SCIROCCO_CYCLE
Gated reset/set/load are not allowed in the design.	A_5C_R_D	DESIGNWARE
	C_1401	LEDA
No gated reset/set/load except in reset/set/load generator RSTGEN.	C_1402	LEDA
	VER_1_3_2_1	VER_STARC_DSG
	VHD_1_3_2_1	VHD_STARC_DSG
Buffers must not be explicitly added to reset/set/load paths.	A_5C_R_C	DESIGNWARE
	C_1403	LEDA
Missing or redundant signals in the sensitivity list of a combinational block. Signal <%item> is missing or redundant.	C_2C_R	DESIGNWARE
	VCS_5	VCS

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Global variable <%item> is referenced in a function.	CS_7C_R_A	DESIGNWARE
	VER_2_1_2_3	VER_STARC_DSG
	W425	VERILINT
Only one clock is allowed in an always block.	S_5C_R_A	DESIGNWARE
	W394	VERILINT
Keyword TRANSPORT ignored in signal assignment.	DCVHDL_2098	DC
	FM_2_27	FORMALITY
	SC_102	SCIROCCO_CYCLE
Internally generated asynchronous reset/set/load <%item> detected.	SC_302	SCIROCCO_CYCLE
	VER_1_3_2_2	VER_STARC_DSG
	VHD_1_3_2_2	VHD_STARC_DSG
Not all cases are covered in full case.	FM_2_12	FORMALITY
	W455	VERILINT
Case choice after the default may be ignored by some simulation tools.	FM_2_18	FORMALITY
	VER_2_8_3_5	VER_STARC_DSG
	W187	VERILINT
Possible range overflow.	FM_2_22	FORMALITY
	VER_2_1_6_4	VER_STARC_DSG
Clock <%item> must be directly controllable from external input port.	TEST_953	DFT
	VER_3_3_1_1	VER_STARC_DSG
	VHD_3_3_1_1	VHD_STARC_DSG
Asynchronous reset/set/load <%item> must be directly controllable from external input port.	TEST_966	DFT
	VER_3_3_1_4	VER_STARC_DSG
	VHD_3_3_1_4	VHD_STARC_DSG

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Tri-state is detected.	DFT_008	DFT
	VER_2_5_1_1	VER_STARC_DSG
	VHD_2_5_1_1	VHD_STARC_DSG
Synchronous reset/set/load <%item> detected.	DFT_017	DFT
	W548	VERILINT
Intra-assignment repeat-event controls for non-blocking assignments are ignored.	DCVER_135	DC
	W300	VERILINT
Infinite recursion detected.	DCHDL_96	DC
	VHD_2_1_8_5	VHD_STARC_DSG
Use a separate line for each HDL statement.	R_526_1	RMM_RTL_CODING_GUIDELINES
	VER_3_1_4_4	VER_STARC_DSG
	VHD_3_1_4_4	VHD_STARC_DSG
The number of characters in one line should not exceed 72 (line <%value>)	G_527_1	RMM_RTL_CODING_GUIDELINES
	VHD_3_1_4_5	VHD_STARC_DSG
Gate instance with too few ports. Port <%format> is not completely connected.	B_1011	LEDA
	DCVER_154	DC
	W287	VERILINT
Only uppercase characters are allowed for parameters.	G_521_3_B	RMM_RTL_CODING_GUIDELINES
	P_3C_G	DESIGNWARE
	VER_1_1_4_2	VER_STARC_DSG
Allocators are not supported for synthesis.	DCVHDL_2100	DC
	SC_029	SCIROCCO_CYCLE

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
No glue logic at top level.	A_8C_R	DESIGNWARE
	C_1004	LEDA
	G_568_1	RMM_RTL_CODING_GUIDELINES
	VER_1_6_4_1	VER_STARC_DSG
	VHD_1_6_4_1	VHD_STARC_DSG
Header, modification, description, date, author, filename field missing from header comment.	R_524_1_B	RMM_RTL_CODING_GUIDELINES
	R_524_1_C	RMM_RTL_CODING_GUIDELINES
	R_524_1_D	RMM_RTL_CODING_GUIDELINES
	R_524_1_E	RMM_RTL_CODING_GUIDELINES
	R_524_1_F	RMM_RTL_CODING_GUIDELINES
Do not use SDF, EDIF or window keywords.	VER_1_1_1_3B	VER_STARC_DSG
	VHD_1_1_1_3B	VHD_STARC_DSG
Hard coded value for bus size is not recommended.	VER_1_1_4_7	VER_STARC_DSG
	VHD_1_1_4_7	VHD_STARC_DSG
Reset names should be RST, plus upto 3 extra characters when multiple resets exists.	VER_1_1_5_2B	VER_STARC_DSG
	VHD_1_1_5_2B	VHD_STARC_DSG
Use asynchronous reset for initial reset to registers.	S_1C_R	DESIGNWARE
	VER_1_3_1_2	VER_STARC_DSG
	VHD_1_3_1_2	VHD_STARC_DSG
Number of I/O ports should be less than 200. This module has <%value> ports.	VER_1_6_4_3	VER_STARC_DSG
	VHD_1_6_4_3	VHD_STARC_DSG

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
LSB of an array should be zero whenever possible.	VER_2_1_6_2	VER_STARC_DSG
	VHD_2_1_6_2	VHD_STARC_DSG
Do not use operators in index of array.	VER_2_1_6_3	VER_STARC_DSG
	VHD_2_1_6_3	VHD_STARC_DSG
Insertion of delay expressions in assignments that infer flipflops is recommended.	VER_2_3_1_3	VER_STARC_DSG
	VHD_2_3_1_3	VHD_STARC_DSG
Use positive integers for delay values in assignments that infer flipflops.	VER_2_3_1_5	VER_STARC_DSG
	VHD_2_3_1_5	VHD_STARC_DSG
If statement in combinational circuit should end with else (not with else if).	VER_2_7_1_3	VER_STARC_DSG
	VHD_2_7_1_3	VHD_STARC_DSG
Number of nested elements in if_statement should be 5 or less.	VER_2_7_3_1	VER_STARC_DSG
	VHD_2_7_3_1	VHD_STARC_DSG
Do not use multiple event control in combinational always block.	VER_2_2_2_3	VER_STARC_DSG
	VHD_2_2_2_3	VHD_STARC_DSG
Using delay values in assignments other than those inferring flipflops are not recommended.	VER_2_3_1_4	VER_STARC_DSG
	VHD_2_3_1_4	VHD_STARC_DSG
Nested event control (@clk1..@clk2) detected. This is not synthesizable.	VER_2_3_3_1	VER_STARC_DSG
	W112	VERILINT
Duplicated case item detected.	VER_2_8_1_3	VER_STARC_DSG
	W547	VERILINT
Do not use defparam.	VER_3_2_4_3	VER_STARC_DSG
	W509	VERILINT
Function descriptions should end with return statements.	VER_2_1_3_4	VER_STARC_DSG
	W489	VERILINT

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Do not change values inside loop variables.	VER_2_9_1_2	VER_STARC_DSG
	W601	VERILINT
In module instantiation, pin name and net name should be the same.	G_521_11	RMM_RTL_CODING_GUIDELINES
	N_8C_G	DESIGNWARE
	VER_1_1_2_6	VER_STARC_DSG
	VHD_1_1_2_6	VHD_STARC_DSG
Missing signal <%item> in sensitivity list.	B_3419	LEDA
	FM_2_1B	FORMALITY
	R_554_1_B	RMM_RTL_CODING_GUIDELINES
	W122	VERILINT
Unequal length LHS and RHS in assignment.	B_3208	LEDA
	FM_2_17	FORMALITY
	VER_2_10_3_2	VER_STARC_DSG
Unequal length port and connexion in module instantiation.	B_3209	LEDA
	VER_3_2_3_2	VER_STARC_DSG
	W110	VERILINT
	XV2_1807	XILINX
	XV2P_1807	XILINX
Unequal length arguments in function call or task enable.	B_3210	LEDA
	VER_2_1_3_1	VER_STARC_DSG
Unequal length between case expression and case item condition in case, casex or casez.	B_3211	LEDA
	VER_2_8_1_6	VER_STARC_DSG
	W263	VERILINT

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
<% value> blocks used to code state machine. One block should be used.	B_3600	LEDA
	CS_1C_R	DESIGNWARE
<% value> blocks used to code state machine. Two blocks should be used.	B_3601	LEDA
	G_559_1	RMM_RTL_CODING_GUIDELINES
	VER_2_11_3_1	VER_STARC_DSG
	VHD_2_11_3_1	VHD_STARC_DSG
Moore style description of state machine is recommended.	B_3602	LEDA
	VER_2_11_1_1	VER_STARC_DSG
	VHD_2_11_1_1	VHD_STARC_DSG
Assign a default state to the state machines.	B_3604	LEDA
	G_559_4	RMM_RTL_CODING_GUIDELINES
Use parameter declarations to define the state vector of a state machine.	B_3605_A	LEDA
	G_559_2_A	RMM_RTL_CODING_GUIDELINES
	VER_2_11_1_3	VER_STARC_DSG
Use an enumerated type to define the state vector of a state machine.	B_3605_B	LEDA
	G_559_2_B	RMM_RTL_CODING_GUIDELINES
Naming convention for state variables. Name should end in _cs.	B_3606	LEDA
	CS_2C_R	RMM_RTL_CODING_GUIDELINES
The number of states in a state machine should be less than 40.	B_3608	LEDA
	VER_2_11_1_4	VER_STARC_DSG
	VHD_2_11_1_4	VHD_STARC_DSG

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
In state machine, keep FSM logic and non-FSM logic apart.	B_3609	LEDA
	VER_2_11_2_1	VER_STARC_DSG
	VHD_2_11_2_1	VHD_STARC_DSG
	G_559_3	RMM_RTL_CODING_GUIDELINES
Flipflop with fixed value data input is detected.	C_1001	LEDA
	VER_2_3_5_1	VER_STARC_DSG
	VHD_2_3_5_1	VHD_STARC_DSG
Signal is used both as synchronous and asynchronous reset/set/load.	C_1404	LEDA
	VER_1_3_1_6	VER_STARC_DSG
	VHD_1_3_1_6	VHD_STARC_DSG
Missing signal <%item> in sensitivity list of combinational always block.	VER_2_2_2_1	VER_STARC_DSG
	VHD_2_2_2_1	VHD_STARC_DSG
Redundant signal <%item> in sensitivity list of combinational always block.	VER_2_2_2_2B	VER_STARC_DSG
	VHD_2_2_2_2	VHD_STARC_DSG
ifdef detected.	VER_3_1_6_1	VER_STARC_DSG
	W529	VERILINT
Flip-flop assigned but not initialized.	B_1403	LEDA
	W396	VERILINT
Latch enabled by a clock feeds latches enabled by the same clock.	TEST_974	DFT
	VER_2_4_1_5	VER_STARC_DSG
	VHD_2_4_1_5	VHD_STARC_DSG
Instance name should be atleast 2 characters.	VER_1_1_2_1C	VER_STARC_DSG
	VHD_1_1_2_1C	VHD_STARC_DSG
Instance name should not exceed 32 characters.	VER_1_1_2_1D	VER_STARC_DSG
	VHD_1_1_2_1D	VHD_STARC_DSG

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Do not use more than one arithmetic operators in one line.	VER_2_10_6_7	VER_STARC_DSG
	VHD_2_10_6_7	VHD_STARC_DSG
Do not use arithmetic operation in the conditional expression of if statement.	VER_2_10_7_1	VER_STARC_DSG
	VHD_2_10_7_1	VHD_STARC_DSG
Latch inferred, make sure that this latch does not have any asynchronous reset/set/load.	VER_2_4_1_3	VER_STARC_DSG
	VHD_2_4_1_3	VHD_STARC_DSG
Tristate buffers connection should not exceed more than 5.	VER_2_5_1_4	VER_STARC_DSG
	VHD_2_5_1_4	VHD_STARC_DSG
Inout should not be directly connected to output.	VER_2_5_1_6	VER_STARC_DSG
	VHD_2_5_1_6	VHD_STARC_DSG
Do not use constant value in the expression of a case_ statement.	VER_2_8_5_2	VER_STARC_DSG
	W226	VERILINT
Case item expression is not constant.	VER_2_8_5_3	VER_STARC_DSG
	W225	VERILINT
Output ports should be registered.	VER_1_6_2_1B	VER_STARC_DSG
	VHD_1_6_2_1B	VHD_STARC_DSG
Active low signals should have suffix _X_N.	VER_1_1_1_7	VER_STARC_DSG
	VHD_1_1_1_7	VHD_STARC_DSG
Do not use module names or instance names that are the same as library cell names.	VER_1_1_1_10	VER_STARC_DSG
	VHD_1_1_1_10	VHD_STARC_DSG
Do not assign 'X' except for the default clause of case statements.	VER_2_10_1_5	VER_STARC_DSG
	VHD_2_10_1_4	VHD_STARC_DSG
Static data types are not supported in \$root.	DC_31	DC
	VCS_31	VCS

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Casting is not supported.	DC_42	DC
	VCS_42	VCS
Import/export of tasks and functions is not supported.	DC_53	DC
	VCS_53	VCS
Process statement is not supported.	DC_54	DC
	VCS_54	VCS
Nested module/interface declaration is not supported.	DC_55	DC
	VCS_55	VCS
Do not use asynchronous reset and asynchronous set in same always block.	VER_1_3_1_7	VER_STARC_DSG
	VHD_1_3_1_7	VHD_STARC_DSG
Delays are ignored by synthesis tool.	FM_2_2	FORMALITY
	W257	VERILINT
Range index out of bound.	DCVER_256	DC
	E267	VERILINT
Name too long for compiled code.	DCHDL_389	DC
	VHD_1_1_2_1B	VHD_STARC_DSG
The number of lines in always statements should not exceed 200.	VER_2_6_1_4	VER_STARC_DSG
	VHD_2_6_1_4	VHD_STARC_DSG
Do not use multiple if statements in the same process statement.	VER_2_6_2_1A	VER_STARC_DSG
	VHD_2_6_2_1A	VHD_STARC_DSG
Do not use multiple case statements in the same process statement.	VER_2_6_2_1B	VER_STARC_DSG
	VHD_2_6_2_1B	VHD_STARC_DSG
Do not use implicit wire declaration.	W154	VERILINT
	VCS_10	VCS

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Do not mix signed and unsigned operands in same operation.	B_3212	LEDA
	VER_2_10_6_3	VER_STARC_DSG
Do not use asynchronous set or reset pins for anything other than initial reset.	NTL_RST05	DESIGN
	VER_1_3_1_3	VER_STARC_DSG
	VHD_1_3_1_3	VHD_STARC_DSG
Limit gate size of a single level to 10000 gates.	NTL_STR99	DESIGN
	VER_1_6_1_1	VER_STARC_DSG
	VHD_1_6_1_1	VHD_STARC_DSG
Hierarchy should contain 2000-10000 gates.	NTL_STR100	DESIGN
	VER_1_6_1_2	VER_STARC_DSG
	VHD_1_6_1_2	VHD_STARC_DSG
Do not use logic in conditional expression to infer tristate.	NTL_STR37	DESIGN
	VER_2_5_1_2	VER_STARC_DSG
	VHD_2_5_1_2	VHD_STARC_DSG
The output of random logic should not be used as a clock.	NTL_CLK12	DESIGN
	VER_3_3_1_3	VER_STARC_DSG
	VHD_3_3_1_3	VHD_STARC_DSG
A clock must not be connected to the D input of a flip-flop.	NTL_STR61	DESIGN
	VER_3_3_3_1	VER_STARC_DSG
	VHD_3_3_3_1	VHD_STARC_DSG
VDD or GND should not be connected to the D input of a flip-flop.	NTL_DFT41	DESIGN
	VER_3_3_3_2	VER_STARC_DSG
	VHD_3_3_3_1	VHD_STARC_DSG

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Do not mix clock and reset lines.	NTL_STR18	DESIGN
	VER_3_3_6_2	VER_STARC_DSG
	VHD_3_3_6_2	VHD_STARC_DSG
A flip-flop output must not be directly connected to an asynchronous set or reset.	NTL_RST06	DESIGN
	VER_3_3_6_3	VER_STARC_DSG
	VHD_3_3_6_3	VHD_STARC_DSG
Tristate enable signals should be controllable from the outside.	NTL_STR54	DESIGN
	VER_3_3_8_2	VER_STARC_DSG
	VHD_3_3_8_2	VHD_STARC_DSG
Gated clocks can only be used at the top level.	NTL_CLK18	DESIGN
	VER_3_4_1_1	VER_STARC_DSG
	VHD_3_4_1_1	VHD_STARC_DSG
Detected internally generated clocks.	NTL_CLK04	DESIGN
	XV2_1203	XILINX
	XV2P_1203	XILINX
Detected input DDR flip-flop.	NTL_STR92	DESIGN
	XV2_1206	XILINX
	XV2P_1206	XILINX
Black box components not in Xilinx DB libraries is detected.	NTL_STR22	DESIGN
	XV2_1614	XILINX
	XV2P_1614	XILINX
Output double data rate register functionality is detected.	NTL_STR93	DESIGN
	XV2_1615	XILINX
	XV2P_1615	XILINX

Table 46: Duplicated Rule List

General Description	Rule Label	Policy
Flip-flip that likely to be merged in synthesis is detected.	NTL_STR85	DESIGN
	XV2_1616	XILINX
	XV2P_1616	XILINX

E**Errors and Warnings Message List****Introduction**

Each table in this appendix lists the compilation and elaboration Error, Warning, Fatal, and Note messages that you may encounter while using Leda.

Verilog Compilation Warnings

Label	Messages
CMPVE004	Unit %s not found in any known library
CMPVE005	Identifier expected after unit name
CMPVE006	Symbol %s is already defined and cannot be redefined as %s
CMPVE007	Symbol %s is not declared in %s %s
CMPVE008	Symbol %s is not defined as %s in %s %s
CMPVE009	Symbol %s has already been defined as input or inout and cannot be redefined as %s
CMPVE010	Unit %s has already been compiled (must be defined twice)
CMPVE011	Unexpected character
CMPVE012	A strength of level %s has already been defined
CMPVE013	Strength value highz1 cannot be used with highz0
CMPVE014	The %s %s declaration should contain at least one input port declaration

Label	Messages
CMPVE015	The symbol %s is not defined as %s
CMPVE016	Global reference of the memory or variable array %s is not allowed
CMPVE017	Operation not allowed on Real
CMPVE018	Integer are limited to 32 bits
CMPVE019	Constant expression required
CMPVE020	Symbol %s has not been defined
CMPVE021	Multiplier of a multiple concatenation must be a constant expression
CMPVE022	Symbol %s is not declared as a function
CMPVE023	Too many arguments in %s
CMPVE024	Too few arguments in %s
CMPVE025	%s is not allowed in a function
CMPVE026	function %s should contain at least one result assignment
CMPVE027	Symbol %s is not declared as a task
CMPVE028	Corresponding port is declared as a Inout or Output so a LValue is expected
CMPVE029	Expression is not a correct lvalue
CMPVE030	Index reference is not allowed on the item %s
CMPVE031	Slice reference is not allowed on the item %s
CMPVE032	The symbol %s has been declared as a memory and cannot drive a port
CMPVE033	A port named %s has already been defined
CMPVE034	Port %s is not declared in the module/primitive definition
CMPVE035	Symbol %s has not been declared as event
CMPVE036	Symbol %s is not a declarative region
CMPVE037	No port named %s is defined in the module %s
CMPVE038	Too many port connection
CMPVE039	Not enough port connection
CMPVE040	Delay control is not allowed in %s instantiation

Label	Messages
CMPVE041	Drive Strength is not allowed in %s instantiation
CMPVE042	Only a delay2 is possible to specify in a primitive instantiation
CMPVE043	The port %s should be an input
CMPVE044	The first port of a UDP must be an output
CMPVE045	Combinational entry is expected in a combinational table
CMPVE046	Sequential entry is expected in a sequential table
CMPVE047	Macro %s is not defined
CMPVE048	Not enough argument to call macro %s
CMPVE049	Too many arguments to call macro %s
CMPVE050	Unexpected endif
CMPVE051	Missing endif
CMPVE052	%s is not a valid default net type
CMPVE053	Invalid argument for unconnected drive
CMPVE054	Invalid time unit
CMPVE055	Duplicate default alternative in case statement
CMPVE056	Port is not connected
CMPVE057	Badly formed identifier, a simple identifier must start with a letter \nUse option \"-k95e\" to accept name starting with '_'
CMPVE058	Slice reference is not allowed on object %s : %s is a scalar object
CMPVE059	No initial statement is allowed in combinational entry
CMPVE060	Attempt to divide by zero
CMPVE061	Badly formed number
CMPVE062	A register must be declared as output in a sequential UDP
CMPVE063	No register must be declared in a combinational UDP
CMPVE064	%s is not allowed in sequential entry
CMPVE065	%s is not allowed in combinational entry
CMPVE066	%s is not allowed in sequential output

Label	Messages
CMPVE067	%s is not allowed in combinational output
CMPVE068	%s is not allowed in sequential current state
CMPVE069	Duplicate entry in table
CMPVE070	Since all inputs of this entry are specified to 'X', the outputs must be specified to "X"
CMPVE071	Only indexed references are allowed on memories
CMPVE072	No module or primitive definition found in input file(s)
CMPVE073	Error in index or range expression : real type expressions are not allowed
CMPVE074	The output of a primitive gate must be a scalar net
CMPVE075	The net or reg should have the same range as the port of the same name
CMPVE076	Illegal reference to block %s: a block name cannot be a used as a terminal reference
CMPVE077	An integer or time port cannot be declared as signed
CMPVE078	Genvar lvalue expected in a genvar assignment statement
CMPVE079	Declarations must precede statements in a named block
CMPVE080	Illegal declaration in anonymous block
CMPVE081	Syntax Error, unexpected token %s in variable declaration
CMPVE082	Syntax Error, unexpected %s in module or udp instantiation
CMPVE083	Syntax Error: %s is already defined in current module or interface declaration
CMPVE084	Syntax Error: received %s while expecting s, ms, us, ns, ps or fs
CMPVE085	Symbol %s is already defined in the current scope
CMPVE086	%s data type is not allowed in packed structure/union
CMPVE087	End name : %s doesn't match the name of the named block : %s
CMPVE088	End name %s doesn't match module/primitive/block name %s
CMPVE089	Use of data type is not allowed in this expression
CMPVE090	Library %s not found

Label	Messages
CMPVE091	Reference to an obsolete version of unit %s appears in the context clause
CMPVE092	Design file is empty
CMPVE093	Unit %s successfully analyzed and stored

Verilog Compilation Failures

Label	Messages
CMPVE094	Design unit %s not found in library %s
CMPVE095	Memory Allocation Failure %s
CMPVE096	Error reading file %s
CMPVE097	Can't open file %s
CMPVE098	Syntax Error
CMPVE099	Preprocessor encountered errors
CMPVE100	Syntax Error
CMPVE101	%s doesn't exist or is not a directory
CMPVE102	Syntax Error : ANSI-C port declaration allowed only in Verilog'2001 & System Verilog mode
CMPVE103	Syntax Error : signed declarations allowed only in Verilog'2001 mode
CMPVE104	Syntax Error : disabling implicit net declaration is allowed only in Verilog'2001 mode
CMPVE105	Maximum macro expansion depth reached, this is probably a recursive macro expansion
CMPVE106	Syntax Error : illegal port type in %s
CMPVE107	Syntax Error : illegal implicit connection mix \"implicit .name\" and \"implicit .*\"
CMPVE108	End of compilation: error(s) in design unit %s
CMPVE109	Maximum number of semantic errors reached. Compilation stopped
CMPVE110	Syntax error : received %s %s

Label	Messages
CMPVE111	Syntax error : %s is not allowed in %s
CMPVE112	Syntax error : %s is allowed only for Verilog2001 or System Verilog mode
CMPVE113	Syntax error : %s is allowed only for System Verilog mode
CMPVE110	Syntax error : received %s
CMPVE111	Syntax error : localparam declaration is not allowed in module parameter port list
CMPVE094	Design unit %s not found in library %s
CMPVE095	Memory Allocation Failure %s
CMPVE096	Error reading file %s
CMPVE097	Can't open file %s
CMPVE098	Syntax Error
CMPVE099	Preprocessor encountered errors
CMPVE100	Syntax Error
CMPVE101	%s doesn't exist or is not a directory
CMPVE102	Syntax Error : ANSI-C port declaration allowed only in Verilog'2001 & System Verilog mode
CMPVE103	Syntax Error : signed declarations allowed only in Verilog'2001 mode
CMPVE104	Syntax Error : disabling implicit net declaration is allowed only in Verilog'2001 mode
CMPVE105	Maximum macro expansion depth reached, this is probably a recursive macro expansion
CMPVE106	Syntax Error : illegal port type in %s
CMPVE107	Syntax Error : illegal implicit connection mix \"implicit.name\" and \"implicit.*\"
CMPVE108	End of compilation: error(s) in design unit %s
CMPVE109	Maximum number of semantic errors reached. Compilation stopped
CMPVE110	Syntax error : received %s %s
CMPVE111	Syntax error : %s is not allowed in %s

Label	Messages
CMPVE112	Syntax error : multi-dimensional arrays allowed only in Verilog2001
CMPVE113	unexpected symbol %s
CMPVE114	Syntax error : %s is not allowed in %s in %s
CMPVE115	Syntax error : %s
CMPVE116	%s is allowed only in \$root module
CMPVE117	Design \"%s\" not found
CMPVE118	%s
CMPVE119	No default working library specified: use the <set> command
CMPVE120	Syntax error : included filename should begin with \" or '<'
LIB001	Intermediate file name too long for design unit %s
CMPVE112	Syntax error : multi-dimensional arrays allowed only in Verilog2001
CMPVE113	unexpected symbol %s
CMPVE114	Syntax error : %s is not allowed in %s in %s
CMPVE115	Syntax error : %s
CMPVE116	%s is allowed only in \$root module
CMPVE117	Design \"%s\" not found
CMPVE118	%s
CMPVE119	No default working library specified: use the <set> command
CMPVE120	Syntax error : included filename should begin with \" or '<'
LIB001	Intermediate file name too long for design unit %s

Deselectable Messages

You can deselect the following message labels using the `rule_deselect` command in the configuration file. For example:

```
rule_deselect -rule WV951019
rule_deselect -rule WV951008
```

Label	Messages
EV951	Index reference is not allowed on the item %s
EV952	Slice reference is not allowed on the item %s
EV953	Global reference is not allowed on the item %s
ESV3	%s statement is legal only within a loop scope
ESV2	"return" statement is legal only within a function or a task
ESV1	Invalid initialization at declaration \n\tNon-module variable \"%s\" cannot be initialized at declaration
ESV2	Property expression cannot be negated twice
ESV3	Illegal use of dollar sign at this place
ESV4	Illegal use of %s before its definition
EV2K2	Illegal assignment of %s at this place
NV950	Unit %s is correct, checked only, not stored in library
NV954	Unit %s is used before it's definition in module %s
NV956	Skipping source library file %s, unable to preprocess it
WV9542	Loaded unit %s.%s is VHDL\%d and current unit is VHDL\%d: inconsistencies may occur...
WV950	The %s %s will be replaced by the %s of the same name !
WV951001	"vectored" or "scalared" keywords are meaningless for the definition of the scalar net %s
WV951002	Drive Strength definition is meaningless because the net %s is not assigned
WV951003	Extra digit in %s : The number has a declared size of %d binary digits and an actual size of %d, the additional left bits are ignored
WV951005	Elements of mintypmax expression should result of the same type

Label	Messages
WV951008	The unit %s is not a verilog unit. Instantiation of a non verilog unit is not supported for the time being
WV951009	Redefinition of macro %s
WV951010	Redefinition of %s %s, only the latest will be compiled
WV951011	Redefinition of the net %s
WV951013	Name is too long for compiled code...
WV951014	No instance name
WV951015	Empty Port in %s
WV951016	%s is a Verilog'2001 keyword and should not be used as an identifier
WV951017	Enumeration label exceeds maximum value
WV951018	not enough port connections for module %s
WV951019	Too many port connection for module %s
WV951020	No port named %s is defined in the module %s
WV951021	Invalid file for -y library : unit name \"%s\" doesn't match file name \"%s\"
CMP0001	Do not access directly an interface object, use the interface through a modport
CMP0002	No type specified for parameter %s
CMP0003	Do not use `timescale directive
CMP0004	Prefix enumerated member with the type name : enum member \"%s\" should be named \"%s_%s\"
CMP0005	The interface object \"%s\" has been declared as output for at least two modports : \"%s\" and \"%s\"
CMP0006	Assertion statement should always have a label
CMP0007	Task/function/net shall be declared before it is used in modports
CMP0008	Enumerated type should be based on specified type
CMP0009	Do not use hard coded values

Elaboration Failure Messages

Label	Messages
ELB004	Library unit %s:%s has not been successfully compiled: so elaboration can not be performed
ELB005	File error format %s not found
ELB006	Memory allocation failure
ELB007	Free memory failure
ELB008	Invalid access to region %d maximum depth :%d !
ELB009	Unit %s of library %s is obsolete versus the elaborator: please recompile unit
ELB010	Unit %s of library %s has a newer version than the elaborator: please get the last elaborator
ELB014	File %s not found to store information about successive elaborated units
ELB015	Value for %s exceeds capacity of external reference
ELB016	External references not yet initialized before use

Elaboration Error Messages

Label	Messages
ELB017	No library with logical name %s
ELB018	No library with physical name %s
ELB019	Unit %s is not a vhdl model
ELB020	Unit %s is not a package
ELB021	Unit %s of library %s is obsolete: it is using more recent unit %s of library %s
ELB022	No unit %s has been found in library %s
ELB023	No entity %s has been found in referenced libraries
ELB024	No architecture found for entity %s in library %s

Label	Messages
ELB025	Unable to resolve %s in %s
ELB026	More than one entity named %s is visible at the same level
ELB027	Maximum number of unresolved instantiation reached: %ld
ELB028	%s in file %s line %d : not yet implemented
ELB029	Trying to elaborate top generic design entities with no generic default value!
ELB030	Trying to connect to object %s with a different type: %s
ELB031	Trying to connect two objects with different kind: %s to %s
ELB032	Different name %s in interface for instantiation of entity %s
ELB033	Unmatched modes for port %s in an instantiation using a default configuration: %s in component, , %s in entity
ELB034	Bad type %s in interface element %s of for instantiation of entity %s
ELB035	Port %s of mode IN unconnected in model %s\n\tof file %s line %ld
ELB036	Expression is equal to %s and is out of the range of the current constraint
ELB037	Type mark %s of object is incompatible with type mark %s of actual
ELB038	Subtype constraint is out of the limits of %s constraint
ELB039	Length of aggregate (= %ld) is greater than length of applicable constraint (= %ld)
ELB040	Length of aggregate (= %ld) is lower than length of applicable constraint (= %ld)
ELB041	Indexed: index %ld is out of range of type %s : bounds %ld and %ld
ELB042	Slice bounds are %ld and %ld : out of range of type %s : %ld and %ld
ELB043	Range length %ld of slice expression is different from range length %ld of its type %s
ELB044	Expression error for %s : trying to divide by zero
ELB045	Expression error : An integer expression cannot be raised to a negative power [LRM.7.2.7]
ELB046	Type conversion error

Label	Messages
ELB047	For type %s conversion, low bound %ld is lower than %ld
ELB048	For type %s conversion, high bound %ld is greater than %ld
ELB049	Error : 64 bits integer too great to be casted into a 32 bits integer:
ELB050	Error : 64 bits real too great to be casted into a 32 bits integer:
ELB051	Unit %s is not a verilog model
ELB052	Primitive %s from library %s can not been elaborated as top unit
ELB053	In library %s unit name %s has not been found. Nevertheless a module %s exists. If it is the wanted unit, please specify it with sensitive case
ELB054	Path is not allowed in \$link_library \"%s\"

Elaboration Warning Messages

Label	Messages
ELB055	Instance %s not fully elaborated (1-%d). Checking might not be completed\n
ELB056	Instance %s not fully elaborated (2-%d). Checking might not be completed\n
ELB057	Instance %s not fully elaborated (3-%d). Checking might not be completed\n
ELB058	Package body %s of library %s is obsolete: package %s of library %s (recompiled after) is elaborated alone without taking body in account
ELB059	Unable to resolve %s in %s
ELB060	Function %s in file %s line %d not yet implemented
ELB061	String length %d is too short to build %s name from concatenation of %s and %s
ELB062	Register %s is illegally connected to output port %s
ELB063	Signal %s has multiple drivers and is not resolved
ELB064	Signal %s between bounds %ld and %ld has multiple drivers and is not resolved

Label	Messages
ELB065	Signal %s has multiple drivers
ELB066	Signal %s between bounds %ld and %ld has multiple drivers
ELB067	Out of bounds: in file %s line %ld:\n\tindex %ld is out of range of object %s : bounds %ld and %ld
ELB068	File %s has not been found among directories of \$search_path: %s
ELB069	Without environment variable \$search_path specified, file %s has been searched in current directory, but not found
ELB070	File %s is not a .db file and can not be taken in account
ELB071	Named association is required for port %s of db cell %s
ELB072	Library %s does not contain functionality for cell %s (%s). LEDA will infer a black-box
ELB073	Number of loops exceeds max limit %ld for verilog generated block %s
ELB073	Number %d of power rails exceeds max limit %ld in %s %s
ELB074	End of elaboration: stack 32 should be empty
ELB075	End of elaboration: stack 64 should be empty
ELB091	Suspicious array %s bound

Elaboration Note Messages

Label	Messages
ELB075	Library %s added with physical path %s
ELB076	Changing path of library %s from %s to %s
ELB077	Too many port connections in instance %s
ELB078	The port %s is of mode IN and has been left unconnected
ELB079	Port %s has bounds left %ld and right %ld, but actual value has a length equal to %ld
ELB080	Port %s has bounds left %ld and right %ld, but actual value with a multiplier equal to %ld has a total length equal to %ld
ELB081	Extra digit in actual binary value %s. : The port %s has a size of %ld binary digits and an actual size of %ld, the additional left bits are ignored
ELB082	Environment variable \$search_path not defined: .db files will only be searched in current directory
ELB083	Environment variable \$link_library not defined: no .db file will be taken in account

Index

Symbols

\$status variable [166](#)
 +checklib command-line option [159](#)
 +define command-line option [159](#)
 +exec+ command-line option [150](#)
 +incdir command-line option [159](#)
 +libtext command-line option [160](#)
 +nochecklib command-line option [157](#)
 +sv command-line switch [160](#)
 +tcl_file command-line option [155](#), [188](#)
 +tcl_rule+ command-line option [156](#)
 +tcl_shell command [188](#)
 +tcl_shell command-line switch [31](#), [156](#),
 [187](#)
 +v2k command-line switch [161](#)
 +v2k switch
 using [65](#)
 .db files [39](#)
 .leda_config.tcl file [73](#)
 .leda_select files [106](#)
 translating [106](#)
 .lib file [39](#)
 .synopsys_leda.setup file [317](#)

A

-a command-line option [159](#)
 Acrobat reader [179](#)
 add_to_collection [197](#)
 all_clocks command [139](#)
 all_inputs [198](#)
 all_inputs command [139](#)
 all_instances [198](#)
 all_outputs [198](#)
 all_outputs command [139](#)
 all_registers [198](#)
 ame [74](#)
 APIs
 C [30](#)

Tcl [30](#)

append_to_collection [198](#)
 Application preferences window [108](#), [111](#)
 Area constraints [138](#)
 Attributes
 defined [26](#)
 max/min [85](#)

B

-b command-line switch [148](#)
 Batch mode [145](#)
 running [148](#)
 Bit blasting [108](#), [111](#)
 -blast switch [148](#)
 -block command-line switch [149](#)
 Block-level checks
 enabling [108](#), [111](#)
 Block-level rule [27](#)
 Browser
 clock [126](#)
 hierarchy [122](#)
 reset [126](#)
 Built-in commands
 Tcl [187](#)

C

C
 API [30](#)
 using [178](#)
 -c command-line switch [149](#)
 -case_analysis command-line option [146](#),
 [149](#)
 -case_analysis option [97](#)
 Cells
 tracing [123](#)
 check [271](#)
 Check boxes
 using [113](#)
 check command [135](#), [136](#), [142](#)

- Check menu [176](#)
- Checker
 - after executing [95](#)
 - command-line [145](#)
 - command-line example [162](#)
 - command-line methods [145](#)
 - command-line results [166](#)
 - common command-line options [148](#)
 - configuring [146](#)
 - control panel [75](#)
 - error viewer [113](#)
 - fixing errors [112](#)
 - invoking [90](#)
 - invoking GUI [170](#)
 - overview [24](#)
 - processing log files [128](#)
 - return status [166](#)
 - running [148](#)
 - saving preferences [108](#)
 - setting preferences [108](#)
 - sorting message display [116](#)
 - using the GUI [169](#)
 - Verilog command-line options [159](#)
 - VHDL command-line options [157](#)
- Checker main window [90, 170](#)
- Checker run options
 - specifying [108, 111](#)
- checker_get_design_constraints command [276](#)
- checker_get_options command [276](#)
- checker_set_design_constraints command [278](#)
- checker_set_options command [280](#)
- chip command-line switch [149](#)
- Chip-level checks
 - enabling [108, 111](#)
- Chip-level rules [27](#)
- Choose a configuration window [99](#)
- Clock and reset tree browsers
 - enabling [110, 126](#)
- Clock Grouping Feature [66](#)
- Clock tree browser [126](#)
- clock_file command-line switch [149](#)
- clockdump command-line switch [126, 149](#)
- Coding rules [27](#)
- Command-line options
 - +checklib [159](#)
 - +define [159](#)
 - +exec+ [150](#)
 - +incdir [159](#)
 - +libtext [160](#)
 - +nochecklib [157](#)
 - +tcl_file [155, 188](#)
 - +tcl_rule+ [156](#)
 - case_analysis [97, 146, 149](#)
 - config [149](#)
 - config_summary [150](#)
 - constraint_file [150](#)
 - f [159](#)
 - files [157](#)
 - html [151](#)
 - i [159](#)
 - ignore_rule_pragmas [151](#)
 - k [159](#)
 - l [151](#)
 - lang [157, 160](#)
 - lappend [151](#)
 - log_dir [151](#)
 - maxhierdump [152](#)
 - maxmessages [152](#)
 - maxviolations [152](#)
 - nochecklib [157](#)
 - noclockdump [152](#)
 - nomaxmessages [153](#)
 - o [153](#)
 - project [153](#)
 - r [154](#)
 - severity [154](#)
 - sort [154](#)
 - summary [155](#)
 - test_asynch [155](#)
 - test_asynch_inverted [155](#)
 - test_clk_falling [155](#)
 - test_clk_rising [155](#)
 - top [156](#)
 - v [161](#)
 - w [161](#)

- work [156](#)
- x [161](#)
- Command-line switches
 - +sv [65](#), [160](#)
 - +tcl_shell [31](#), [156](#), [187](#)
 - +v2k [65](#), [161](#)
 - a [159](#)
 - b [148](#)
 - blast [148](#)
 - block [149](#)
 - c [149](#)
 - chip [149](#)
 - clock_file [149](#)
 - clockdump [126](#), [149](#)
 - d [159](#)
 - forcehierdump [150](#)
 - full_inf [150](#)
 - full_log [126](#), [151](#)
 - h [151](#)
 - mk [157](#)
 - mkk [157](#)
 - netlist [152](#)
 - nobanner [152](#)
 - nocode [152](#)
 - nocompilemessage [152](#)
 - noecho [152](#)
 - nohierdump [153](#)
 - nolog [153](#)
 - nomaxviolations [153](#)
 - nowarning [153](#)
 - old_format [153](#)
 - q [160](#)
 - quiet [154](#)
 - s [160](#)
 - sdc [135](#), [154](#)
 - summary [129](#)
 - sverilog [65](#), [160](#)
 - t [160](#)
 - translate_directive [156](#)
 - u [160](#)
 - upgrade400 [106](#), [156](#)
 - use_netlist_reader [161](#)
 - uselrmsize [160](#)
 - usev2klrmsize [161](#)
 - version_directive [156](#)

- Commands
 - elaborate [188](#)
 - export [86](#)
 - gui_start [31](#)
 - gui_stop [31](#)
 - import [86](#)
 - leda [90](#), [148](#), [170](#)
 - leda -specifier [80](#), [170](#)
 - setup_custom [316](#), [318](#)
- compare_collections [199](#)
- config command-line option [149](#)
- config_summary command-line option [150](#)
- Configuration
 - files [72](#)
 - search path for rules [74](#)
- Configurations
 - Custom [99](#)
 - Default [99](#)
 - Gate-level [99](#)
 - global checking [74](#)
 - Leda-classic [99](#)
 - Leda-optimized [99](#)
 - loading default [99](#)
 - loading saved [73](#)
 - prebuilt [99](#)
 - restoring [73](#)
 - RTL [99](#)
 - sdc-equivalency [99](#)
 - sdc-postlayout [99](#)
 - sdc-prelayout [99](#)
 - sdc-quality-postlayout [99](#), [388](#)
 - sdc-quality-prelayout [99](#), [390](#)
 - sdc-quality-rtl [99](#), [393](#)
 - sdc-rtl [99](#)
 - sdc-top-versus-block [99](#)
 - using [99](#)
- Configuring rules [102](#)
- connect_power_domain [200](#)
- Constant detection
 - block-level [49](#)
- Constant propagation
 - in batch mode [97](#)
 - in GUI mode [97](#)
 - in Tcl shell mode [97](#)

Constants
 propagating 96
 Constraint file 97
 Constraint Query Language 136
 -constraint_file command-line option 150
 -constraint_file option 150
 Conventions
 documentation 21
 typographical and symbol 21
 copy_collections 200
 CQL 136
 create_clock command 137
 create_generated_clock command 137
 create_operating_conditions 197, 199
 create_power_domain 200
 create_power_net_info 201
 current_design command 139, 282
 Custom rules
 compiled 72
 directory location 72

D

-d command-line switch 159
 Deactivating rules
 from Error Viewer 105
 from HDL files 103
 with .synopsys_leda.setup file 101
 delete_operating_conditions 201
 Design Compiler 39, 133
 Design queries
 fast track 188
 Design Query Language 30
 Design report 120
 Design rule constraints 137
 Design rules
 about 39
 Rules
 design 28
 Designs
 checking for errors 89
 Detecting sets/resets 49
 disable_isolation_cell_recognition 202
 Disabling Redundant Rules 399

Displays
 file 119
 rule 117
 DQL 30
 Duplicated Rules
 error messages 399
 view redundant rules 399

E

Editor
 Leda default 172
 selecting 172
 Vi 172
 XEmacs 172
 elaborate 188
 elaborate command 135, 188, 283
 enable_isolation_cell_recognition 202
 Environment
 checking 171
 Environment variables
 checking settings 171
 HTML_NAVIGATOR 318
 LEDA_CLOCK_FILE 318
 LEDA_CONFIG 72, 74, 318
 LEDA_HTML_DOC_PATH 318
 LEDA_HTML_USR_PATH 318
 LEDA_LANGUAGE 115, 318
 LEDA_MAX_CLOCKS 318
 LEDA_PATH 72, 86, 318
 LEDA_READER 318
 LEDA_RESOURCES 147, 316, 319
 LEDA_RULES 72
 LEDA_SELECT_FILE 319
 link_library 40, 319
 LM_LICENSE_FILE 319
 search_path 40, 319
 setting 317
 SNPSLMD_LICENSE_FILE 319
 Error report
 saving 127
 Error Viewer
 using check boxes 113
 Error viewer 113
 configuring 116

- displays [117](#)
- preferences window [116](#)
- sorting and filtering [117](#)
- Errors
 - fixing [112](#)
- Errors and Warnings Messages [433](#)
- Examples
 - Checker command-line [162](#)

F

- f command-line option [159](#)
- File menu [173](#)
- Files
 - .db [39](#)
 - .leda_select [106](#)
 - .lib [39](#)
 - .synopsys_leda.setup [317](#)
 - constraint [97](#)
 - leda.inf [167](#), [172](#)
 - leda.log [128](#), [129](#), [154](#)
 - leda_command.log [188](#)
 - leda_config.tcl [73](#)
 - leda_history.log [111](#)
 - log [128](#)
 - makefile [314](#)
 - PDF [179](#)
 - plibs [146](#)
 - project [91](#)
 - ruleset.rl [81](#)
 - ruleset.sl [81](#)
 - SDC [133](#)
- files command-line option [157](#)
- filter_collection [202](#)
- Finite state machines [42](#)
 - inferring [42](#)
 - Mealy [42](#)
 - Moore [42](#)
- forcehierdump command-line switch [150](#)
- foreach_in_collection [203](#)
- FSM [42](#)
- full_inf command-line switch [150](#)
- full_log command-line switch [126](#), [151](#)

G

- Gate-level prebuilt configuration [325](#)
- Generating Log Files in Batch Mode [163](#)
- Generating Log Files in GUI Mode [186](#)
- Generating Log Files in Tcl Mode [307](#)
- Get top module/design entity window [94](#)
- get_all_input_boundaries_from_power_domain [203](#)
- get_all_output_boundaries_from_power_domain [203](#)
- get_cells [210](#)
- get_cells command [139](#)
- get_clocks [204](#)
- get_clocks command [139](#)
- get_lib_cells command [139](#)
- get_lib_pins command [139](#)
- get_libs command [139](#)
- get_nets [204](#)
- get_nets command [139](#)
- get_nth_power_net [205](#)
- get_object_name [205](#)
- get_pins [207](#)
- get_pins command [139](#)
- get_ports [208](#)
- get_ports command [139](#)
- get_power_cells [205](#)
- get_power_domains [208](#)
- get_power_down [206](#)
- get_power_down_ack [206](#)
- get_power_net_max_voltage [206](#)
- get_power_net_min_voltage [206](#)
- get_power_net_source_port [207](#)
- get_power_net_type [207](#)
- getn_power_net [207](#)
- Getting help [22](#)
- GUI
 - invoking [170](#)
 - saving preferences [108](#)
 - setting references [108](#)
- GUI mode [169](#)
- gui_start command [31](#)
- gui_stop command [31](#)

H

- h command-line switch [151](#)
- Hardware Inference [43](#)
- Hardware rules [27](#)
- Hardware semantics
 - defined [27](#)
 - Verilog [27](#)
 - Verilog example [27](#)
 - VHDL [27](#)
 - VHDL example [27](#)
- Hardware-based rules
 - about [42](#)
 - finite state machines [42](#)
- HDL source files
 - managing from GUI [181](#)
- Help
 - for Tcl commands [189](#)
 - getting tool support [22](#)
- Help menu [178](#)
- HTML
 - help file [78](#)
- html command-line option [151](#)
- HTML_NAVIGATOR environment variable [318](#)

I

- i command-line option [159](#)
- ignore_rule_pragmas command-line option [151](#)
- index_collection [209](#)
- infer_power_domain [209](#)
- infer_power_domains [209](#)
- Info report [171](#)
- Invoking Checker GUI [170](#)
- Invoking GUI [170](#)
- Invoking Specifier GUI [170](#)
- is_64bit [197](#)

J

- Japanese
 - prepackaged rule help [115](#)

K

- k command-line option [159](#)

L

- l command-line option [151](#)
- lang command-line option [157, 160](#)
- lappend command-line option [151](#)
- Leda
 - about [180](#)
 - batch mode [30](#)
 - changing modes [188](#)
 - defined [23](#)
 - GUI mode [30](#)
 - how it works [25](#)
 - how to use [29](#)
 - invoking in Tcl shell mode [187](#)
 - modes of operation [30](#)
 - on the Web [180](#)
 - overview [24](#)
 - overview diagram [24](#)
 - switching modes [30, 31](#)
 - Tcl shell mode [30](#)
 - terminology [29](#)
 - types of rules [27](#)
 - using batch mode [30](#)
 - using GUI mode [30](#)
 - using Tcl shell mode [30](#)
 - using the GUI [169](#)
 - version info [180](#)
 - what is it? [23](#)
- leda.inf file [167, 172](#)
- leda.log file [128, 129, 154](#)
- LEDA_CLOCK_FILE environment variable [318](#)
- leda_command.log file [188](#)
- LEDA_CONFIG environment variable [72, 74, 318](#)
- leda_history.log file [111](#)
- LEDA_HTML_DOC_PATH environment variable [318](#)
- LEDA_HTML_USR_PATH environment variable [318](#)
- LEDA_LANGUAGE environment variable [115, 318](#)

LEDA_MAX_CLOCKS environment variable [318](#)

LEDA_PATH environment variable [72](#), [318](#)

LEDA_READER environment variable [318](#)

LEDA_RESOURCES environment variable [147](#), [316](#), [319](#)

LEDA_RULES environment variable [72](#)

LEDA_SELECT_FILE environment variable [319](#)

Leda-classic prebuilt configuration [327](#)

Leda-optimized prebuilt configuration [327](#)

Libraries

- adding to VHDL resource projects [315](#)
- building [314](#)
- golden [93](#)
- logical/physical mapping [146](#)
- managing units from GUI [184](#)
- mapping [146](#)
- setting VHDL [313](#)
- setting VHDL resource [314](#)
- specifying [93](#)

Library pop-up menu [182](#), [186](#)

Library unit manager window [184](#)

link command [285](#)

link_library environment variable [40](#), [319](#)

LM_LICENSE_FILE environment variable [319](#)

Load configuration window [78](#)

Local VHDL resource libraries, creating [316](#)

Log files

- printing summary info [154](#)
- sorting [154](#)

-log_dir command-line option [151](#)

Logic assignments [138](#)

M

Macros

- advanced programming [85](#)

Make

- solving problems [314](#)

Makefile [314](#)

Managers

- library unit [184](#)

Manual

- VRSL Reference Guide [179](#)

Manuals

- Language Reference Manuals (LRMs) [51](#)
- Leda C Interface Guide [178](#)
- Leda Installation Guide [178](#)
- Leda Release Notes [178](#)
- Leda Rule Specifier Tutorial [19](#), [178](#)
- Leda Tcl Interface Guide [178](#)
- Verilog LRM [51](#)
- VerSL Reference Guide [39](#), [179](#)
- VHDL LRM [51](#)

Mapping libraries [146](#)

Max violations

- default [100](#) [109](#)
- setting [109](#)

-maxhierdump command-line option [152](#)

-maxmessages command-line option [152](#)

-maxviolations command-line option [152](#)

Menus

- check [176](#)
- file [173](#)
- help [178](#)
- library pop-up [182](#), [186](#)
- project [175](#)
- project pop-up [182](#)
- report [177](#)
- source file pop-up [183](#)
- unit pop-up [183](#), [186](#)
- view [178](#)
- window [178](#)

Mixed-language designs

- writing and checking [59](#)

-mk command-line switch [157](#)

-mkk command-line switch [157](#)

Modes

- batch [145](#)
- changing [188](#)
- GUI [169](#)
- Tcl shell [187](#)

Modules/Units

- pop-up windows [185](#)

N

Netlist checks [30](#)
 enabling [108](#), [111](#), [188](#)
 -netlist command-line switches [152](#)
 Netlist queries
 fast track [188](#)
 Netlist Reader [66](#), [68](#)
 Netlist rules [28](#)
 Nets
 tracing [123](#)
 -nobanner command-line switch [152](#)
 -noclockdump command-line option [152](#)
 -nocode command-line switch [152](#)
 -nocompilemessage command-line switch [152](#)
 -noecho command-line switch [152](#)
 -nohierdump command-line switch [153](#)
 -nohierdump switch [135](#)
 -nolog command-line switch [153](#)
 -nomaxmessages command-line option [153](#)
 -nomaxviolations command-line switch [153](#)
 -nowarning command-line switch [153](#)

O

-o command-line option [153](#)
 -old_format command-line switch [153](#)
 Opening Projects
 in batch mode [32](#)
 in GUI mode [32](#)
 in Tcl shell mode [32](#)
 Operating condition [137](#)
 Options
 Verilog [92](#)
 VHDL [92](#)

P

Path Mill [39](#)
 Path View
 changing schemas [125](#)
 Path Viewer

extended [124](#)
 extending schematic [125](#)
 hierarchy browser [122](#)
 linking to source code [123](#)
 scanning to primary ports [125](#)
 scanning to sequentials [125](#)
 see full hierarchical name [123](#)
 setting preferences [124](#)
 standalone [124](#)
 tracing backward [123](#)
 tracing forward [123](#)
 traversing hierarchy [125](#)
 using [121](#)
 Path Viewer window [121](#)
 PDF file reader
 Acrobat [179](#)
 Pins
 tracing [123](#)
 plibs file [146](#)
 global [147](#)
 local [147](#)
 syntax [147](#)
 Policies
 defined [26](#)
 exporting [86](#)
 importing [86](#)
 prepackaged [75](#)
 Verilint [104](#)
 Policy manager window [80](#), [81](#)
 Porosity constraints [138](#)
 Power constraints [138](#)
 Pragmas
 leda off and leda on [103](#)
 synthesis_off and synthesis_on [42](#)
 translate_off and translate_on [42](#), [92](#)
 verilint off and verilint on [104](#)
 Prebuilt configurations
 Gate-level [325](#)
 Leda-classic [327](#)
 Leda-optimized [327](#)
 RTL [322](#)
 SDC-prelayout [390](#)
 SDC-RTL [393](#)
 Preferences

- saving [108](#)
- setting [108](#)
- PrimeTime [133](#)
- print_config_summary [210](#)
- project command-line option [153](#)
- Project creation wizard [313](#)
- Project file
 - creating [91](#)
- Project menu [175](#)
- Project pop-up menu [182](#)
- Project pop-up window [185](#)
- project_add_library command [253, 254](#)
- project_delete command [255](#)
- project_get_all_files command [255](#)
- project_get_file_attributes command [256](#)
- project_get_library_attribute command [257](#)
- project_get_option_attribute command [258](#)
- project_get_ports command [258](#)
- project_get_top_units command [259](#)
- project_get_unit_kinds_from_library command [259](#)
- project_get_units_from_file command [260](#)
- project_get_units_from_library command [261](#)
- project_get_working_libraries command [262](#)
- project_new command [262](#)
- project_open command [263](#)
- project_quit command [263](#)
- project_read command [264](#)
- project_record_cmd command [264](#)
- project_remove_file command [265](#)
- project_remove_library command [265](#)
- project_save command [266](#)
- project_specify_files command [266](#)
- project_specify_libraries command [267](#)
- project_specify_name command [268](#)
- project_specify_options command [269, 276, 278, 282](#)
- project_update command [269](#)
- Projects
 - creating [91](#)

- creating mixed-language [165](#)
- generating in batch mode [163](#)
- updating [130](#)
- propagate command [140, 141, 286](#)
- Propagating constants [96](#)

Q

- q command-line switch [160](#)
- query_objects [210](#)
- quiet command-line switch [154](#)

R

- r command-line option [154](#)
- read_constraints command [136, 140, 287](#)
- read_files command [289](#)
- read_sverilog [293](#)
- read_sverilog command [293](#)
- read_verilog command [135, 141, 296](#)
- read_vhdl command [135, 141, 299](#)
- Regular expressions [83](#)
- Related documents [19](#)
- remove_clock_gating_cell [210](#)
- remove_from_collection [210](#)
- remove_isolation_cell [211](#)
- remove_level_shifter [211](#)
- remove_power_domain [211, 212](#)
- remove_power_net_info [212](#)
- remove_voltage_domain [212](#)
- report command [136](#)
- Report menu [177](#)
- report_clock_gating_cells [212](#)
- report_enable_pin [213](#)
- report_isolation_cells [213](#)
- report_level_shifter [213](#)
- report_operating_conditions [214](#)
- report_pin_voltage [214](#)
- report_power_domains [214](#)
- report_power_net_info [215](#)
- report_power_pins [215](#)
- report_power_switches [216](#)
- Reports
 - design [120](#)

- error [117](#)
- info [171](#)
- summary [116](#)
- Reserved Variables [307](#)
- Reset detection [49](#)
- Reset tree browser [126](#)
- reset_isolation_cell_recognition [216](#)
- RTL prebuilt configuration [322](#)
- Rule configuration file
 - config.tcl [102](#)
- Rule selection files
 - translating [106](#)
- Rule wizard
 - configuring [73](#)
 - locked wizard warning [76](#)
 - locking [75](#)
 - selecting configuration [99](#)
- Rule wizard window [74, 98](#)
- rule_deselect command [135](#)
- rule_get_all_masters_from_topic command [219](#)
- rule_get_all_rules_from_master_id command [220](#)
- rule_get_all_topics command [221](#)
- rule_get_configuration command [222](#)
- rule_get_current_configuration command [223, 236](#)
- rule_get_parameter command [142, 217](#)
- rule_get_policies command [224](#)
- rule_get_policy_attributes command [225](#)
- rule_get_predefined_configurations command [226](#)
- rule_get_rules command [227](#)
- rule_get_ruleset_attributes command [228](#)
- rule_get_rulesets command [229](#)
- rule_get_selection command [218](#)
- rule_get_templateset_attributes command [230](#)
- rule_get_templatesets command [231](#)
- rule_link command [232](#)
- rule_load command [232](#)
- rule_load_configuration command [233](#)
- rule_manage_policy command [234](#)

- rule_patch command [235](#)
- rule_save_configuration command [235](#)
- rule_select command [135, 239](#)
- rule_set_default_configuration command [237](#)
- rule_set_html command [240](#)
- rule_set_message command [241](#)
- rule_set_parameter command [142, 241](#)
- rule_set_predefined_configuration command [238](#)
- rule_set_severity command [247](#)
- Rules
 - block-level [27](#)
 - chip-level [27](#)
 - choosing creation method [79](#)
 - coding [27](#)
 - configuring [71, 102](#)
 - configuring prepackaged [74](#)
 - creating [71, 79](#)
 - creating new [79](#)
 - deactivating [101](#)
 - defined [26](#)
 - design [39](#)
 - Disabling Redundant Rules [101](#)
 - hardware [27](#)
 - methods for creating [79](#)
 - modifying [71](#)
 - netlist [28](#)
 - SDC [28](#)
 - types [27, 146](#)
 - types Leda cannot check [50](#)
 - writing from scratch [80](#)
- Ruleset
 - defined [26](#)
- ruleset.rl file [81](#)
- ruleset.sl file [81](#)
- run command [301](#)

S

- s command-line switch [160](#)
- SDC [110](#)
 - object names [139](#)
- SDC checker
 - defining parameters for rules [142](#)

- policy 135
- prepackaged rules 135
- setting environment variables 136
- simplified usage model 135
- Tcl commands 140
- using 133
- using Tcl script 141
- SDC checks
 - enabling 108, 111
- sdc command-line switch 135, 154
- SDC file
 - error handling 140
 - setting version 141
 - supported commands 137
- SDC rules 28
- sdc_apply command 141, 303
- sdc_set command 140
- SDC-postlayout Prebuilt Configuration 388
- SDC-prelayout prebuilt configuration 390
- SDC-RTL prebuilt configuration 393
- search_path environment variable 40, 319
- Semantic exceptions
 - defined 51
 - enabling or disabling 92
 - Verilog 57
 - Verilog examples 57
 - VHDL 52
 - VHDL examples 52
- Set default text editor window 172
- Set detection 49
- set_case_analysis 97, 149
- set_case_analysis command 138, 304
- set_clock_gating_cell 247
- set_clock_latency command 137
- set_clock_transition command 137
- set_clock_uncertainty command 137
- set_data_check command 137
- set_disable_timing command 137
- set_drive command 137
- set_driving_cell command 137
- set_enable_pin 248
- set_false_path command 138
- set_fanout_load command 137
- set_gating_clock_check command 137
- set_input_delay command 137
- set_input_transition command 137
- set_level_shifter 248
- set_load command 137
- set_logic_dc command 138
- set_logic_one command 138
- set_logic_zero command 138
- set_max_area command 138
- set_max_capacitance command 137
- set_max_delay command 138
- set_max_dynamic_power command 138
- set_max_fanout command 137
- set_max_leakage_power command 138
- set_max_time_borrow command 137
- set_max_transition command 137
- set_min_capacitance command 137
- set_min_delay command 138
- set_min_fanout command 137
- set_min_porosity command 138
- set_multicycle_path command 138
- set_operating_conditions 248
- set_operating_conditions command 137
- set_output_delay command 137
- set_pin_voltage 249
- set_port_fanout_number command 137
- set_power_domain 250
- set_power_domain_ctrl 250
- set_power_off_value 250
- set_power_pin 249, 253
- set_power_switch 250, 251, 253
- set_propagated_clock 137
- set_resistance command 137
- set_voltage_domain 253
- set_wire_load_min_block_size command 137
- set_wire_load_mode command 137
- set_wire_load_model command 137
- set_wire_load_selection_group command 137
- setup_custom command 316, 318
- severity command-line option 154

- Signals
 - supply0 97
 - supply1 97
 - sizeof_collection 251
 - SNPSLMD_LICENSE_FILE environment variable 319
 - sort command-line option 154
 - sort_collection 251
 - Source file manager window 181
 - Source file pop-up menu 183
 - Source files
 - specifying 94
 - Specifier
 - as compiler 25
 - building rules 72
 - main window 80
 - overview 24
 - using the GUI 169
 - Specifier GUI
 - invoking 170
 - Specify design information window 109
 - Specify files window 94
 - Specify libraries window 93, 313
 - Specify project window 91
 - Standards
 - SystemVerilog 65, 93
 - Verilog 2001 65, 93
 - Verilog 95 93
 - VHDL 87 52
 - VHDL 93 52
 - summary command-line option 155
 - summary command-line switch 129
 - Summary report 116
 - Support
 - Leda 22
 - SystemVerilog 65
 - Verilog 2001 65
 - sverilog command-line switch 160
 - sverilog switch
 - using 65
 - Switching modes 31
 - Synopsys Design Constraints
 - checking 110
 - Synopsys Design Constraints checker 133
 - SYNOPSIS pragmas 92
 - synthesis_off pragma or directive 42, 156
 - synthesis_on pragma or directive 42, 156
 - System interface 137
 - SystemVerilog
 - enabling 65
- ## T
- t command-line switch 160
 - Tcl
 - API 30
 - built-in commands 187
 - checker commands 271
 - getting command help 189
 - invoking shell mode 187
 - project commands 253
 - rule commands 197
 - syntax rules 133
 - using 178
 - Tcl Shell mode
 - with SDC checks 135
 - Tcl shell mode 187
 - Templates
 - defined 26
 - Templatesets 72
 - Terminology
 - Leda 29
 - Test clock/reset window 110
 - Test mode
 - propagating constants 96
 - test_async command-line option 155
 - test_async_inverted command-line option 155
 - test_clk_falling command-line option 155
 - test_clk_rising command-line option 155
 - Text editor
 - Leda default 172
 - selecting 172
 - Vi 172
 - XEmacs 172
 - Timing constraints 137
 - Timing exceptions 138
 - Tools

- Checker [23](#)
- Specifier [23](#)
- top command-line option [156](#)
- Tracing
 - backward [124](#)
 - forward [124](#)
- translate_directive command-line switch [156](#)
- translate_off pragma or directive [42, 92, 156](#)
- translate_on pragma or directive [42, 92, 156](#)
- Tutorials
 - Leda rule specifier [178](#)

U

- u command-line switch [160](#)
- Unit pop-up menu [183, 186](#)
- UNIX
 - regular expressions [83](#)
- Update the project window [130](#)
- upgrade400 command-line switch [106, 156](#)
- use_netlist_reader command-line switch [161](#)
- User environment
 - checking [171](#)
- usev2klrmsize command-line switch [161](#)
- Using Regular Expressions with Hierarchy [193](#)

V

- v command-line option [161](#)
- VCS [56](#)
- verify command [305](#)
- Verilint
 - policy [104](#)
 - pragmas [104](#)
 - rules [104](#)
- Verilog
 - instantiating in VHDL [63](#)
 - mapping VHDL identifiers [64](#)
 - modules [56](#)

- UDPs [56](#)
- Verilog 2001 [65, 93](#)
- Verilog 95 [93](#)
- Verilog designs
 - writing and checking [56](#)
- Verilog types
 - mapping to VHDL [62](#)
- version command-line switch [156](#)
- VerSL
 - defined [27](#)
- VHDL
 - adding files to resource projects [315](#)
 - compilation order [315](#)
 - design entities [52](#)
 - instantiating Verilog units [64](#)
 - loads and resets [49](#)
 - mapping Verilog identifiers [64](#)
 - resource libraries [52, 93](#)
 - semantic exceptions examples [53](#)
 - setting libraries [313](#)
 - setting resource libraries [314](#)
 - working libraries [93](#)
- VHDL 87 [92, 314](#)
- VHDL 93 [92, 314](#)
- VHDL data types
 - mapping to Verilog types [60](#)
- VHDL design entities
 - in Verilog modules [60](#)
- VHDL designs
 - writing and checking [52](#)
- VHDL semantic exceptions [52](#)
- Vi [172](#)
- View menu [178](#)
- VRSL
 - defined [27](#)
- vuselrmsize command-line switch [160](#)

W

- w command-line option [161](#)
- Warning
 - lock rule wizard [76](#)
- Window menu [178](#)
- Windows

- application preferences [108, 111](#)
- checker control panel [75](#)
- checker main [90, 170](#)
- choose a configuration [99](#)
- clock and reset tree browser [126](#)
- configure prepackaged rules [77](#)
- error viewer preferences [116](#)
- get top module/design entity [94](#)
- info report tab display [172](#)
- library unit manager [184](#)
- load configuration [78](#)
- path viewer [121](#)
- policy manager [80, 81](#)
- project pop-up [185](#)
- rule wizard [74, 98](#)
- set default text editor [172](#)
- source file manager [181](#)
- specify design information [109](#)
- specify files [94](#)
- specify libraries [93, 313](#)
- specify project [91](#)
- test clock/reset [110](#)
- update the project [130](#)
- Wire load models [137](#)
- Wizard [313](#)
 - configuring [74](#)
- work command-line option [156](#)

X

- x command-line option [161](#)
- XEmacs [172](#)