

nLint

User Guide and Tutorial

Synopsys, Inc.

Version

This manual supports nLint 2013.09 and higher versions. You should use the documentation from the version of the installed software you are currently using. This manual was printed on August 27, 2013.

Copyright and Proprietary Information Notice

(c) 2013 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.

Third-Party Software Notices

nLint includes or is bundled with software licensed to Synopsys under free or open-source licenses. For additional information regarding Synopsys's use of free and open-source software, refer to the *third_party_notices.txt* file included within the <INSTALL_PATH>/doc directory of the installed nLint software.

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <http://www.synopsys.com/Company/Pages/Trademarks.aspx>.

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

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

Contents

About This Book	1
Overview.....	1
Audience	1
Book Organization.....	1
Conventions Used in This Book.....	2
Related Publications	3
Introduction	5
Overview.....	5
nLint Command Line Options (Batch Mode).....	6
nLint Graphical User Interface (GUI Mode).....	6
nLint Rule Groups	8
nLint Benefits	9
Quick Start	11
Quick Start: an nLint Tutorial.....	11
Clock Source Tree Extraction Tutorial	25
Overview.....	25
User Interface	39
Overview.....	39
File Commands	42
Import Design.....	42
Save File.....	45
Save As.....	45
Export Options	46
Restore Session	46
Save Session.....	46
Load Report DB	47
Save Report DB.....	47
Save Report DB as	47
Suppress	47
File Order	48

Exit	48
Edit Commands	49
Undo	49
Redo	49
Cut	49
Copy	49
Paste	50
Delete	50
Select All	50
Find	50
Find Next	52
Replace	52
Advanced	52
Bookmarks	53
Run Commands	54
Project Setting	54
Create Domain	73
Compile	74
Lint	74
Local Lint	74
Tools Commands	75
Find Scope	75
Test Regular Expression	76
Debussy	76
Rule Organizer	76
Report Viewer	77
Preferences	77
Violation Command	92
Next Violation Line	92
Previous Violation Line	92
Next Violation	92
Previous Violation	93
Window Command	94
Output Window	94
Help Commands	94
Contents	94
Rules Reference	94
About	94

Rule Organizer **95**

Overview	95
----------------	----

File Commands	97
Open	97
Save	99
Save As.....	99
Export Rules to CSV File.....	99
Export Rules	99
Properties.....	101
Recent Files	101
Close.....	101
Group Commands	102
Enable.....	102
Non-configurable	102
Rename.....	102
New Top Group.....	102
Set As Active Top Group.....	102
Add Sub Group	103
Copy	103
Paste	103
Delete	103
Rules Commands	104
Copy	104
Paste	104
Delete	104
Properties.....	104
Map Severity	108
Find Rules	109
Select All.....	111
Invert Selection	111
Enable/Disable	111
Translate Off/On	111
Status non-configurable	111
Argument non-configurable	112
View Commands	113
Sort	113
Ascending.....	113
Descending.....	113
Language	113
Administration Mode/Lint Mode	113
Preferences	114
Window Panel.....	114
Help Commands	114
Contents.....	114

Rules Reference	114
Search Rule	115
About.....	115

Report Viewer 117

Overview.....	117
Lint from Main Frame Window (GUI Mode).....	117
Lint from the Shell Prompt (Batch Mode)	118
File Commands	119
Open	119
Save	119
Save As.....	119
Close.....	119
Export Violations	119
Export Clock Domains	121
Close Window	121
View Commands	122
Sort By	122
Filter	124
Skip Filter.....	127
Skip Mark.....	128
Toggle Mark.....	128
Find	128
Find Next.....	129
Go To	129
Forward	130
Backward.....	130
Add See Also.....	130
Domain Commands	131
Specify It As Source.....	131
Find Register	131
Find String.....	131
Hide	132
Expand.....	133
Move To	134
Tools Commands	135
Disable Rule	135
View Error Source.....	135
Preferences	135
Commands in Shortcut Menu	136
Violation.....	136

Windows Panel	138
Help Command	138
Search Rule	138

Pragmas 139

Overview.....	139
//novas Pragmas	139
Syntax.....	139
Semantics	139
Usage Notes	140
Examples	141
//synopsys Pragmas.....	142

Command Line Options 145

Command Line Syntax	145
Option -actualwidth/-actualwidth_off	152
Option -logdir	152
Option -logfile	152
Option -top top_module.....	153
Option -lintTop entry_scope.....	153
Enable/Disable a rule in command line -r/+r.....	153
Enable/Disable a rule in command line -pr/+pr.....	154
Option -dm module.....	154
Option -sm module_name.....	154
Option -bb module_name	155
Option -df file/-uf file_list	155
Detail message -detail.....	155
Option -opt filename.....	156
Option -vf format_string.....	156
Option -ignore_pragma.....	156
Option for Partition.....	157
Option -partition_top.....	157
Option -partitioin_module.....	157
Option -partition scope.....	157
Option to specify clock source	157
Option for Beautify Output File	158
Option for Line Compiler Directive	159
Option -filter filter_file.....	159
Option for Return Value	160
Option -on_rule_exit All <rule_number> AnyCompilerError	160
Option -ee exit_value	160

Option -active_group top_group.....	160
Option to create clock domain and source tree.....	161
Option -vclk_source <name>.....	161
Option -clk_source.....	161
Option -clk_uncern signal.....	162
Option -gated_clk_cell <cell.port>.....	162
Option -clk_lasr on/off.....	162
Option -clk_masr on/off.....	162
Option to extract crossing path.....	163
Option -clk_export filename.....	163
Option -clk_export_reg.....	164
nLint Return Value	167
Overview.....	167
nLint Return Values.....	167
Rule Groups	169
Overview.....	169
Design Issues.....	169
Readability Issues.....	170
Rule Setting File	171
Overview.....	171
RS File Format.....	171
RS File Version.....	171
Top Groups.....	172
Group Settings.....	172
Parameter for Special Rule.....	174
Compiler Errors.....	176
Merge Criteria.....	177
Top Groups Section.....	177
Default Group Section.....	177
Group Include Section.....	177
Rules In Group.....	178
Special Rule Parameter.....	179
Suppress a Violation	181
Overview.....	181
Set Up Lint Entry.....	181
Disable/Enable a Rule.....	181

Initial All Rules Enable/Disable Statically	181
Import Design.....	182
Resolving Rule Disable/Enable by Design Context.....	183
Traversing Design	184
Checking	184
UDR Checking.....	185
Filter Out Violations	185
Filter File Format	186
Applying Filter Setting.....	187

Clock Source Tree 189

Overview.....	189
Terminology	189
Clock Signal	189
Internal Clock Source.....	189
Clock Domain	189
Clock Source Tree.....	190
Clock Source	192
Methods To Extract Clock Source Tree.....	193

About This Book

Overview

This book is designed to allow you to quickly become proficient in nLint, the comprehensive HDL Design Rule Checker. The book should be read from beginning to end. Sections you are already familiar with can be skipped.

Audience

The audience for this book includes chip designers who require faster and more efficient automated and customizable checking tools.

This document assumes that you have a basic knowledge of the platform on which your version of nLint runs: Unix, Linux, or Windows NT; and that you are knowledgeable in Verilog/VHDL and digital logic design.

Book Organization

This *nLint User Guide and Tutorial* is organized as follows:

- “About This Book” provides an introduction to this book and explains how to use it.
- “Installing nLint” lists supported platforms and explains how to install nLint and to set up the associated files before starting nLint.
- “Introduction” provides an overview of nLint, the comprehensive design rule checking system.
- “Quick Start” guides you step-by-step through nLint checking process.
- “Clock Source Tree Extraction Tutorial” guides you step-by-step through clock analysis.
- “User Interface” describes the graphical user interface and primary windows, including Project Window, Editor Window and Output Window.
- “Rule Organizer” explains the menu commands in Rule Organizer.
- “Report Viewer” explains the menu commands in Report Viewer.

- “Pragmas” describes two types of pragmas supported in nLint.
- “Command Line Options” covers the command line syntax to invoke nLint.
- “Rule Groups” explains the design and readability issues of rule groups.
- “Rule Setting File” provides the file format of rule setting (RS) file, which is used to configure the checking rules and arguments for each rule.
- “Suppress a Violation” describes all the aspects that will affect the report of a violation in nLint.
- “Clock Source Tree” describes the concept of clock domain.
- “Index” is the index to this book.

Conventions Used in This Book

The following conventions are used in this book:

- *Italics font* is used for emphasizes, book titles, section names, design names, file path and file names within paragraphs.
- **Bold font** is used to emphasize menu items, function keys, button name, i.e., **Create Path** command, press **F2** to save, click **OK** button, etc; **Menu -> Command** identifies the path used to select a menu command.
- *Blue text* is used to outline text area for active hypertext links; it helps you jump to the reference topic.
- `Courier type` is used for program listings, TCL command and arguments. It is also used for test message that displays on the screen. For instance:
- **Typed in Text** is used to indicate a paragraph text that a user needs to type in. For example, type **filename** and then press **Return**.
- **NOTE** describes important information, warnings, or unique commands.
- Left-click or Click means click the left mouse button on the indicated item.
- Middle-click means click the middle mouse button on the indicated item.
- Right-click means click the right mouse button on the indicated item.
- Double-click means click twice consecutively with the left mouse button.
- Shift-left-click means press and hold the <Shift> key then click the left mouse button on the indicated item.
- Shift-right-click means press and hold the <Shift> key then click the right mouse button on the indicated item.
- Drag-left means press and hold the left mouse button, then move the pointer to the destination and release the button.

Related Publications

- *nLint Rule Category*
Gives detailed description on nLint checking rules, provided with rule message, rule group information and reference of Verilog/VHDL coding example.
- *nLint Guide to User-definable Rules and Open KDB Reference Manual*
Provides guide to user-definable rules (UDR) and Open KDB Reference.
- *nLint Release Notes and Installation Files*
For current information about the latest software version, see *nLint Release Notes* shipped with the product and the *Installation Files* in the distribution directories.

Introduction

Overview

nLint is a HDL design rule checker that can help hardware designers to create syntax and semantics correct HDL code. Specifically, nLint reads in HDL source code, analyzes it, and outputs warnings and errors (including error position and error messages) in a report viewer. Through this comprehensive checking scheme, nLint automatically identifies coding errors, such as asynchronous loop, gated clock, RTL and gate level simulation mismatches, etc., which might otherwise get carried unnoticed into simulation, synthesis or ATPG stages. In other words, nLint can catch errors at design entry time, thus preventing the designers from wasting time in finding and correcting errors, especially those careless mistakes, at a later design stage. This can significantly speed up the design process.

nLint enforces naming conventions and coding styles to improve the source code readability and maintainability. In the SOC era when IC design routinely involves a team of designers and reusable components (IP's) are not uncommon, it is very important that designers can read each other's code with relative ease. In this regard, nLint is an invaluable tool.

Meanwhile, nLint also verifies clock domain in the design. It automatically extracts clock domains in the design and lists all crossing paths between different domains. Based on the knowledge of clock domain, nLint may check those clock domain related rules.

nLint checks against copious amount of RMM compliant* rules. In detail, nLint supports about 300 semantic rules for Verilog and 500 rules for syntax checking; meanwhile, it also supports about 300 semantic rules for VHDL and about 230 rules for syntax checking.

nLint is a self-contained EDA tool. It does not require a simulator, synthesizer, or any other EDA tools license to run.

nLint supports User-defined Rules (UDR) checking. Based on the TCL interface, user can write procedures to query design information and to check design rules.

nLint provides a complete user-friendly environment for error viewing and correction. User can use the built-in editor to edit the source code while viewing the errors. After editing, user can save the edited source code and re-run nLint to

verify that those errors have indeed been fixed. These checking, viewing and correcting tasks can all be done efficiently in the GUI environment that nLint provides.

You may execute nLint by the following modes:

- a GUI mode, which provides a user-friendly interface for various tasks, such as linting, viewing error, etc.
- a batch mode, which performs the rule checking and reports errors and warnings in file.

** Reuse Methodology Manual For System-On-Chip Designs, a Synopsys/Mentor's book on IP re-use.*

nLint Command Line Options (Batch Mode)

By using nLint's batch mode, you may also take input from the command line, perform the rule checking, and generate an output. To invoke, you need to specify the source code files to be checked, the output file to save the result, and other command line options if needed.

See [Command Line Options](#) for more details.

nLint Graphical User Interface (GUI Mode)

nLint provides a user-friendly GUI mode for the following services:

- Import design and list all source files, (Project Window)
- Enable or disable rules for checking and organize the rule set by your favorite, (Rule Organizer)
- Specify which part of the design files are to be skipped for checking, (Project Window)
- Start rule checking action, and
- View and correct errors

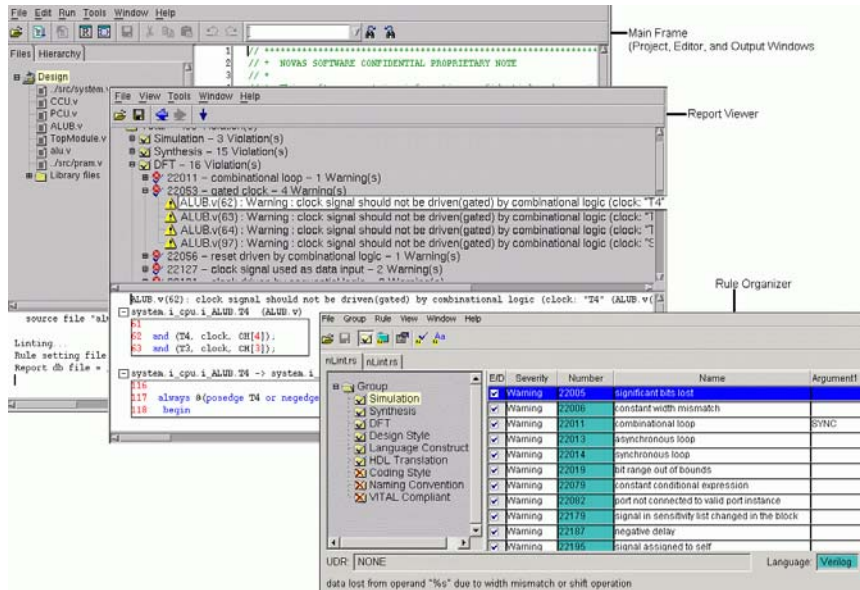


Figure: nLint Main Frame, Report Viewer and Rule Organizer

Before invoking the rule checking action, you need to import the source code files in Project window. Next, invoke Rule Organizer to configure the checking rules by clicking the **Rule Organizer** button or by selecting the **Tool -> Rule Organizer** command. Then, start rule checking by clicking the **Lint Design** button or by selecting the **Lint -> Run** command.

During the checking process, a progress window appears, reporting the checking progress. You may halt the process by clicking the **Stop** button available in the progress window if needed. When the checking process is complete, the results will be shown in a report viewer.

For more information, refer to [User Interface](#).

nLint Rule Groups

For easy understanding and managing, rules checked by nLint are categorized as in the following:

Rule groups concerning the use of EDA tools

- Simulation
- Synthesis
- DFT
- VITAL Compliant

Rule groups concerning the design and coding practices

- Design Style
- Language Construct
- HDL Translation
- Clock

Rule groups concerning the readability

- Naming Convention
- Coding Style

NOTE: These rules may appear in more than one category. For example, the rule to check for the combinational feedback loops is under the **Design Style** rule group (it's not a good design style) as well as the **DFT** rule group (it cannot be handled by most ATPG tools.)

Rule categories and rules are summarized and explained later in this manual. Detailed descriptions for each rule can be found in the *nLint Rule Category*.

nLint Benefits

Besides helping create syntax and semantics correct HDL code, nLint can be used to

- ensure a robust design style, such as synchronous design, homogeneous clocking scheme and set/reset signals, etc.
- encourage the usage of certain language constructs while discourage certain others,
- enforce a coding style and naming convention among development team members.

In summary, nLint can

- shorten the time the designer will need to spend in various EDA tools, such as simulator, synthesizer, ATPG tools, etc.,
- uncover potential errors at an early stage, thus reducing the need for design cycle iteration,
- create readable and maintainable code to facilitate collaborative design, and
- achieve the goal of design re-use.

Quick Start

Quick Start: an nLint Tutorial

In this tutorial, we will use a simple design as an example to guide you through the major features in nLint. Steps we will go through are as follows:

- Specify the rules to be checked.
- Specify the UDR scripts.
- Specify the design file (using a *run.f* file).
- Invoke the nLint checking process.
- View and edit errors.

View source code in the built-in editor

- Jump to the source code that causes the errors.
- Edit the source code to correct the error.
- Save the file and re-run nLint to verify that the error has been corrected.

View source code in the Verdi system

- Set source code display to *nTrace/nSchema* automatically.
- Double-click a violation and *nTrace* will display the relevant source code.
- Double-click a violation and *nSchema* will display the relevant path.

1. Invoke nLint Graphical User Interface (GUI)

Before we start, we need to get a fresh copy of the sample source code, and make sure that you have write privilege on those files. Please go to the directories:

```
<nLint_install_dir>/demo/verilog
```

And issue the following commands:

```
> cp -rf original/* .
> chmod 777 *
> cd rtl
```

Now, let's start nLint GUI by typing the following command at the shell prompt:

```
> nLint -gui
```

The main frame window will be shown on the screen:

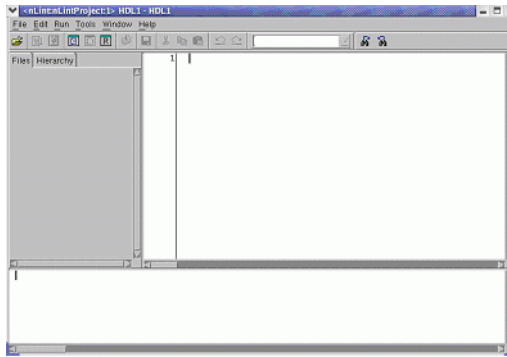


Figure: nLint Main Frame Window

There are three sub-windows in the main frame window: Project Window, Editor Window and Output Window.

2. Specify the Design File

Then, we need to specify the design file by using the **File -> Import Design** command. After the command is invoked, a form with the **From Library** and **From File** tabs will be opened. For this tutorial, we will use the **From File** method. Click on the **From File** tab, an *Import Design* form appears as shown below. You can use this to browse the directories and files on the disk. Please change the directory to `<nLint_install_dir>/demo/verilog/rtl` and select the `run.f` file, as shown below:

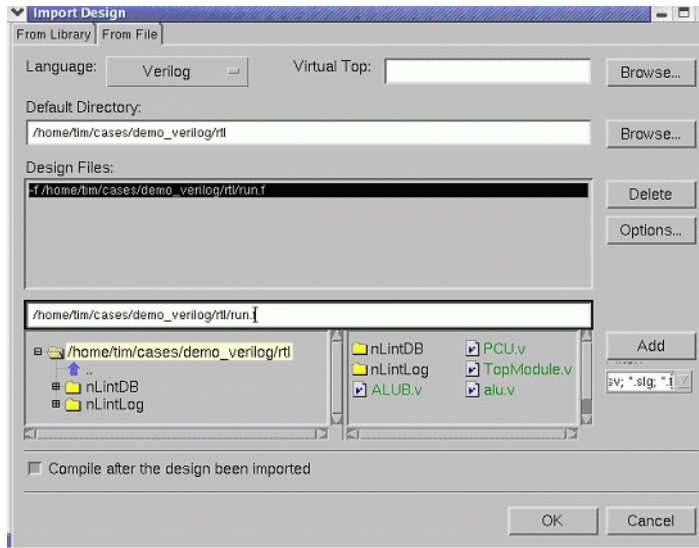


Figure: File -> Import Design -> From File

Click the **OK** button to import the design. By default, nLint will compile the design when import the design files. Then, the file names of the design file will be listed in the Project Window as shown. Notice that the first file is automatically shown in the integrated Editor Window.

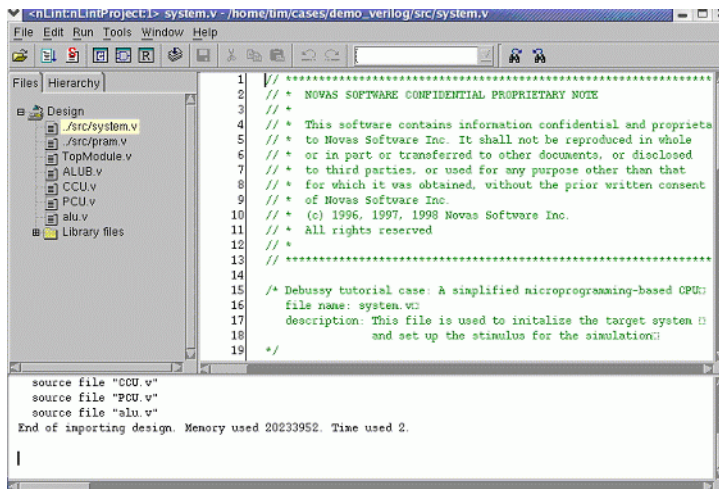


Figure: Show File Content in Editor Window

3. Edit more than one file at the same time

When you double-click on a file, the file contents will be shown in the Editor Window. Clicking the right mouse button on a file displays a shortcut menu with an **Open** command and Suppress File Command. Select the **Open** command. The selected file will be shown in another editor window, as illustrated below.

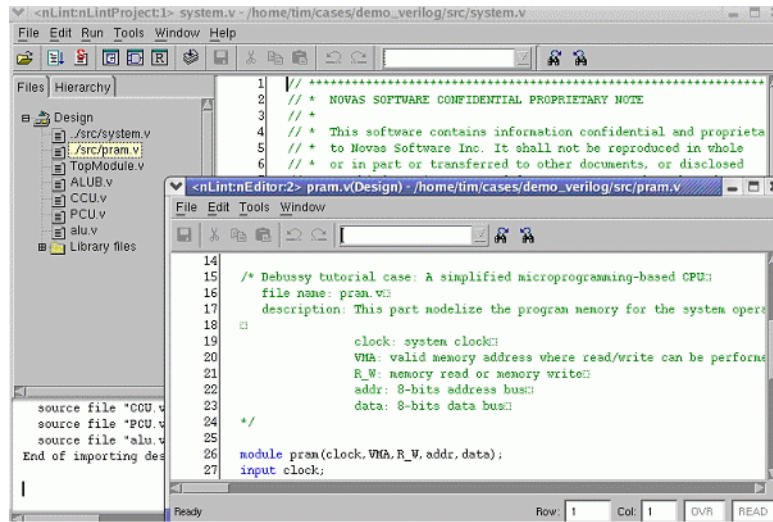


Figure: Show File in Another Window

4. Configure to suppress some files

Select a file and click on the **Suppress** button. An unchecked icon will be attached on the file if the status of the command is unchecked.

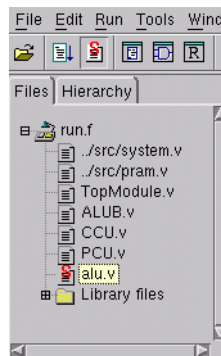


Figure: File is Attached with Unchecked Icon

5. Configure to check from a sub-design

Go to the **Hierarchy** tab, the design hierarchy tree is shown. Select a node and click-right to choose the **Lint from here** command, which specifies the checking process to lint the sub-design from *system.i_cpu*.

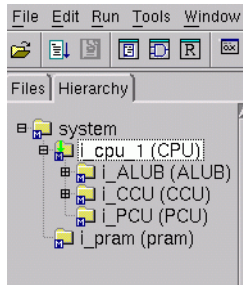


Figure: Lint from *system.i_cpu*

6. Invoke Rule Organizer

Then, we need to start the rule organizer by issuing the **Tools -> Rule Organizer** command. There are two major windows in Rule Organizer: the one to the left displays a list of groups, and the one to the right displays a list of rules under the selected group.

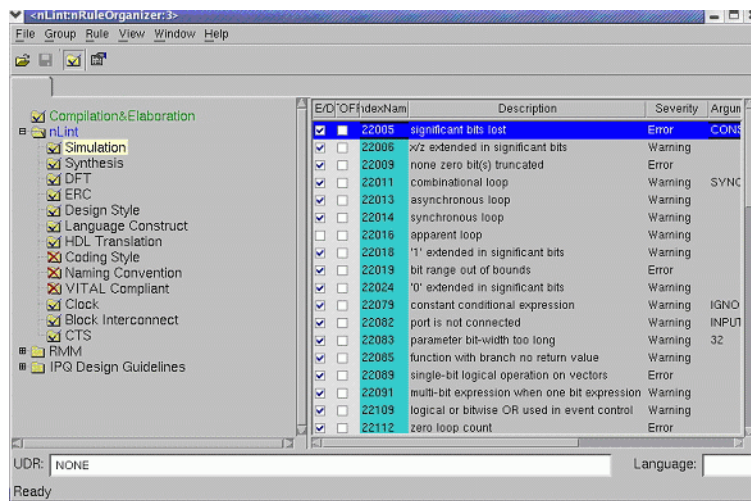


Figure: Rule Organizer

There are three top groups, **Compilation & Elaboration**, **nLint**, and **RMM**.

The **Compilation & Elaboration** group, which cannot be removed or modified, includes the rules for syntax checking.

The nLint group includes the following ten sub-groups: Simulation, Synthesis, ERC, DFT, Design Style, Language Construct, HDL Translation, Naming Convention, Coding Style, VITAL Compliant, Clock, Block Interconnect and CTS. By default, the Naming Convention, Coding Style, VITAL Compliant and CTS sub-groups are disabled, i.e., rules in those three sub-groups are not checked by default. This is indicated by a red X marked on the icon of these three sub-groups. (If for some reasons the default setting has been changed in your environment, you can use the **Groups -> Enable** command to disable these three sub-groups.)

The **RMM** group includes the groups and rules that are named by the *Reuse Methodology Manual For System-On-Chip Designs*, a book on IP re-use.

The IPQ Design Guidelines group includes the groups and rules that are defined by SoC Technology Center of Taiwan, China.

In the above figure, the active top group is nLint, which is marked with blue color. The **Compilation & Elaboration** group will always be included in checking regardless of which is the top group.

You can get help on a rule by selecting that rule in Rule Organizer and pressing the **F1** key (Note that the rules in **Compilation & Elaboration** group are not supported currently). After that, if you select a different rule in Rule Organizer, you will need to press **F1** again in order to make the online help to jump to that rule.

7. UDR Scripts

By default, under the *demo* directory of `<nLint_install_dir>`, there is an *udr* directory, in which there are two files, *udr.ini* and *action.tcl*, for UDR scripts. If you use `nLint -gui-udr <nLint_install_dir>/demo/udr` to invoke nLint, the UDR scripts from this directory will be loaded. The user-definable rules (UDR) specified in the *udr.ini* file are categorized in a **User Defined** group and these rules will be added into the active top group and be shown in the Rule Organizer, as illustrated below.

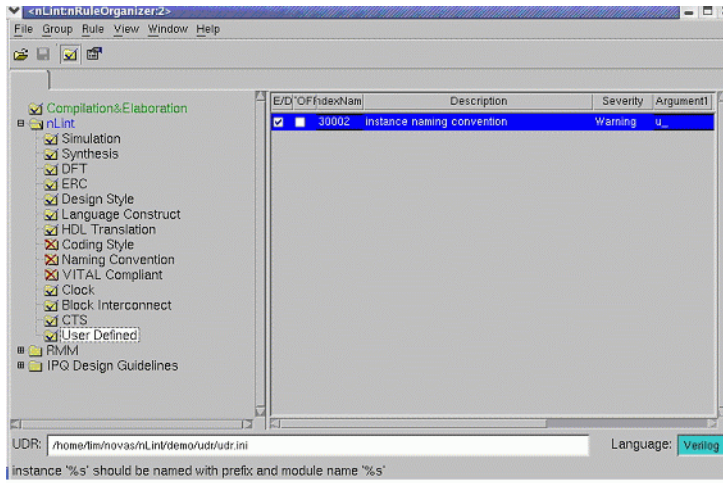


Figure: UDR in Rule Organizer

There is one rule in the *udr.ini* file. Rule 30002 is to check if the instance name should have a prefix (*u_*) plus the name of the defined module.

8. Start Checking

In the main frame window, select **Run -> Lint** to start the checking process. While running the process, a progress window appears, as shown below.

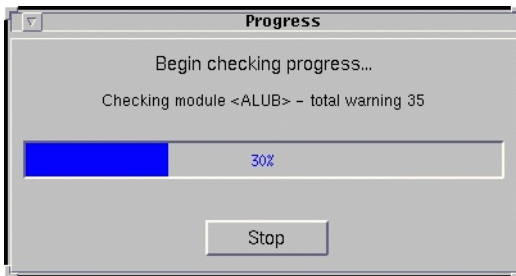


Figure: Show Checking Progress

The progress bar shows approximately how much work has been done, and how much more to go. You can click the **Stop** button to stop the checking process at any time.

9. Viewing and Editing Violations

After the checking process is done, the result will be reported in a Report Viewer, which consists of two sub-windows: Tree and Detail Windows.

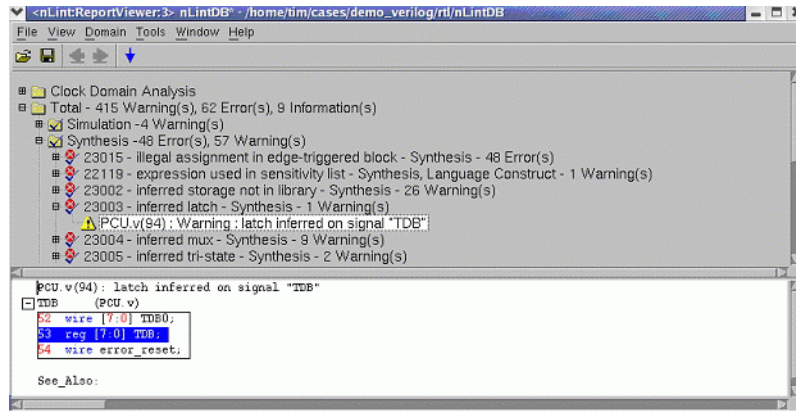


Figure: Report Viewer is Displayed After Checking Process

The reported violations are sorted first by groups and then by rules; they are shown in a tree structure at the upper part of the review viewer. You can click on the plus (+) icon to expand the rule, thus showing more details, or click on the minus (-) icon to collapse, thus hiding the details.

When left-clicking on a rule message, the detailed information will be shown in the detail window, the lower part of the report viewer, with three sections: the first section is the violation itself; the second is the source code in quick view, enclosed with a rectangular box; and the third section (See Also) lists the related violations.

For this tutorial, please expand the **Simulation** rule group, and double-click on the first rule under that group. Reported messages under that rule is shown as following.

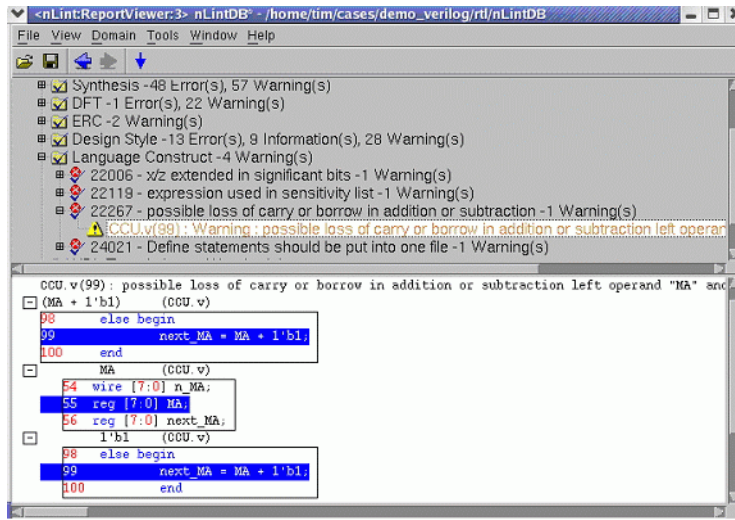


Figure: Report Viewer - Simulation Rule Group is Expanded

When clicking on the violation 22283, the detailed information is shown in the detail window, in which you can find the module definition with port declaration, notice that there is a port `addr_error`. You can also find the module instance `i_maprom`, notice that there is no port instance to the port `addr_error`.

In the tree window of report viewer, double-click on the violation, `CCU.v(78)`, the Editor Window in the main frame window will open the source file and jump to the related source code line and highlight it as follows:

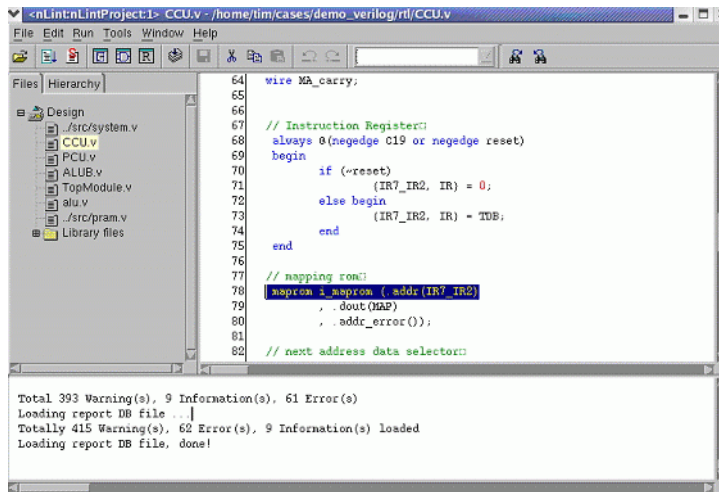


Figure: Show Related Source Code in Editor Window

You can see that the error is about the open connection on the `addr_error` instance port. Let's just assume that we can ignore that for now.

Next, let's expand the **Language Construct** rule group. You will see that the first violation is about *clock driven by expression*.

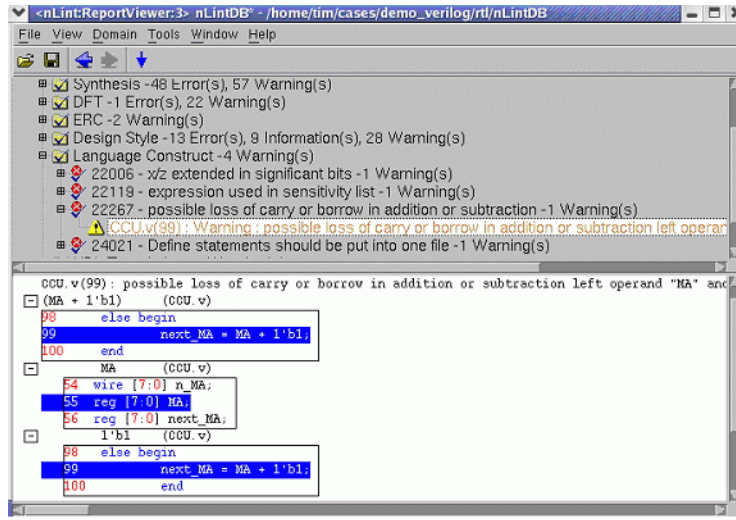


Figure: Report Viewer - Language Construct Rule Group Is Expanded

Double click on the violation message, `CCU.v(99)`, and you will see the following figure in the Editor Window:

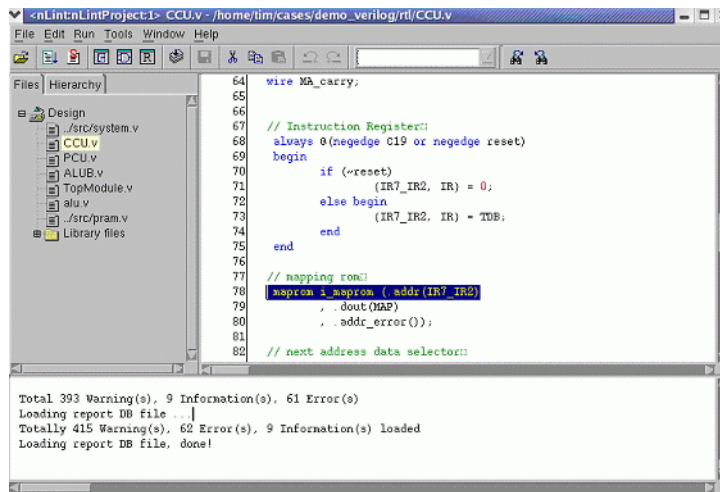


Figure: Show Related Source Code in Editor Window

This is about the assignment ($\text{next_MA} = \text{MA} + 1'b1$). The carry bit of an addition is possible to get lost in assignment. Let's use the editor to modify the code by adding a carry signal, such as a value: `MA_carry`, in the left hand side of the assignment. See following figure for explanation.

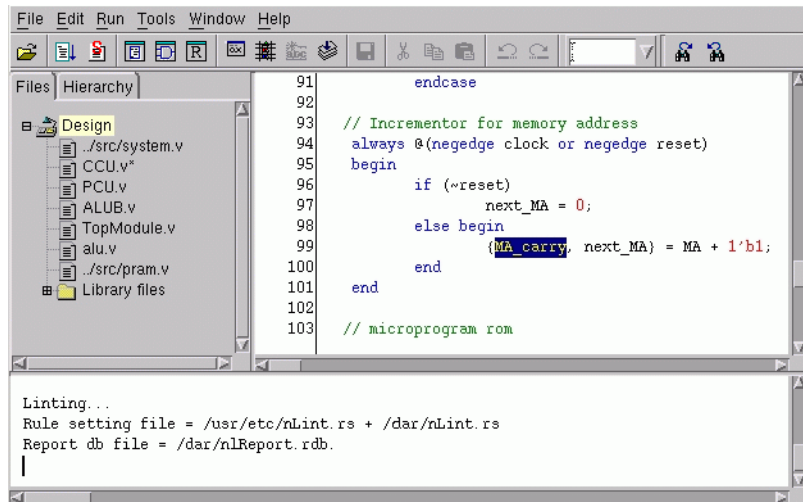


Figure: Modify Source Code with nLint Editor

In addition, you may also need to modify the declaration of `MA_carry`, as shown below:

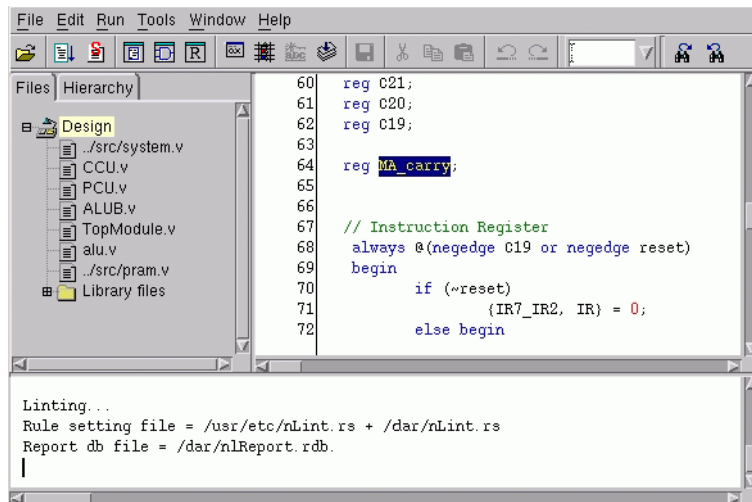


Figure: Modify Declaration of MA_carry

10. Save Edited Source File and Re-run the Checking Process

Now, we can save the edited source code, and re-run the checking (by re-issuing the **Run -> Lint** command) to verify that the error has indeed been fixed.

11. Set Source Display to *nTrace/nSchema*

If you are a Verdi user, you can link nLint to *nTrace/nSchema*. This integration between nLint and *nTrace/nSchema* is very useful since many types of errors reported by nLint involve multiple parts of the source code. To analyze these types of errors, you need the powerful tracing and viewing capabilities from the Verdi system.

To set up, you need to use the **Tools -> Preferences** command in nLint. A *Preferences* form opens as shown below:

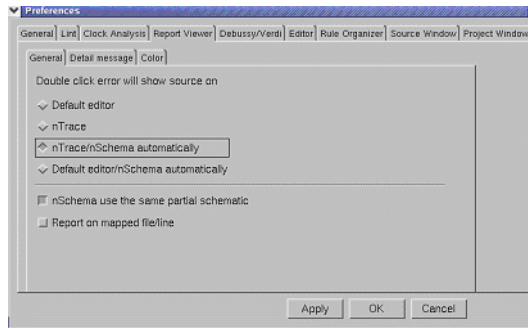


Figure: Preferences Form

In the **Report -> General** tab, there are four options under **Double click error will show source on**: **Default Editor**, **nTrace**, **nTrace/nSchema automatically** and **Default Editor/nSchema automatically**. The **Default Editor** option is enabled by default, nLint will display the source code in the built-in Editor Window. Choosing the **nTrace/nSchema automatically** option causes nLint to show the relevant source code in the suitable *nTrace/nSchema* window. For example, the information of some violations, i.e. 21001, *signal name should be upper or lower case*, is suitable to show source code in *nTrace*, while the information of some violations, i.e. 22011, *combinational loop*, is suitable to show source code in *nSchema*.

When the preferences are set, click **OK** to close the form. nLint will save the new settings in the *nLint.rc* file in your working directory. The next time you launch nLint, it will use the last settings saved in this *nLint.rc* file.

Next, let's go back to the report viewer in nLint and expand the DFT rule group. There is violation of rule 22053, which is a gated clock. Double-click on the

violation reported at ALUB.v(97), you will see that *nSchema* window is invoked, and the path from the gate output to the clock port of the register is shown in the *nSchema* window. Following is the figure.

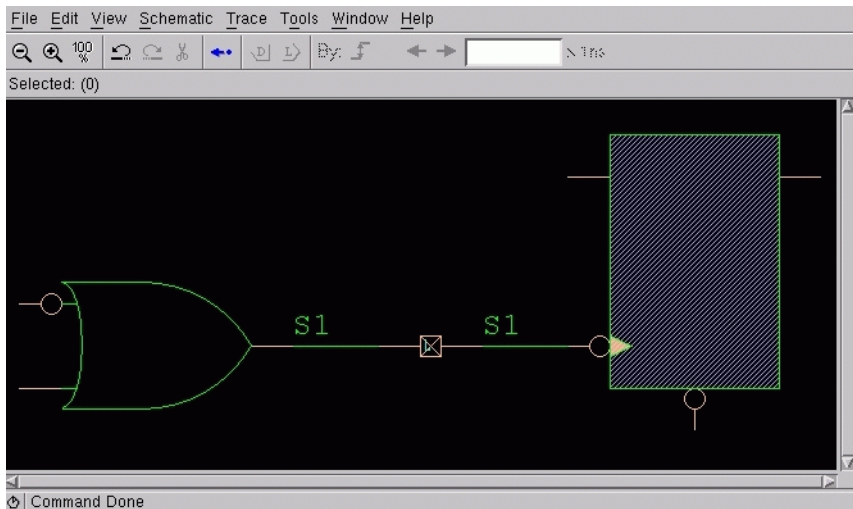


Figure: Gated Clock Error is Shown in *nSchema* Window

At this point, we can invoke the *nTrace* window to display the source code of the gate inferred and the register inferred, and drag the *or* gate over (from *nSchema* window). In the *nTrace* window, we can see clearly about the inferred *or* gate.

```

88         2:   Y0 = 8'h00;
89         3:   Y0 = 8'h00;
90         4:   Y0 = PC;
91         5:   Y0 = IXR_tmp;
92         6:   Y0 = ACC_tmp;
93         default: Y0 = 8'h00;
94     endcase
95
96     assign net_1 = (IR[0] == 0) ? zero_flag : carry_flag;
97     assign S1 = "CH[0] | ("CH[1] & (IR[1] ^ net_1));
98
99     always @(negedge clock or negedge reset)
100     begin
101         if (!reset) IXR_tmp = 8'h00;

```

source file "TopModule.v"
source file "ALUB.v"
source file "CCU.v"
source file "PCU.v"
source file "alu.v"
Linking... 0 error(s), 0 warning(s)
Total 0 error(s), 0 warning(s)

Figure: Inferred OR gate is shown in *nTrace* Window

Then you can drag the register over. The source code inferred by the register will be shown in *nTrace* window as below:

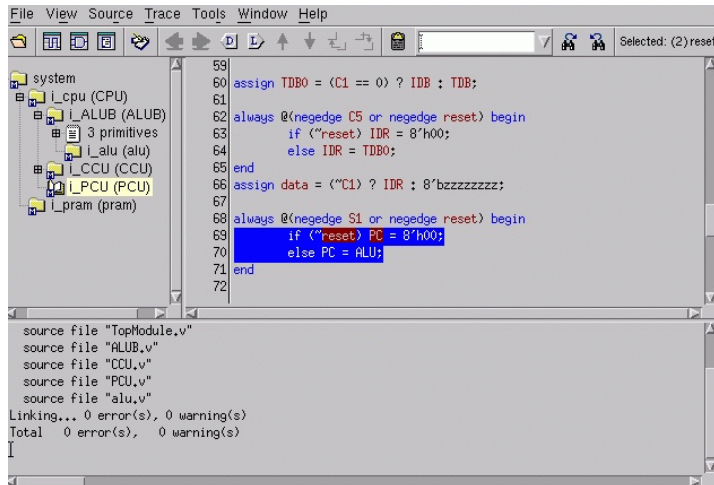


Figure: Inferred Flip-Flop Register is Shown in *nTrace* Window

If you want to change back to the nLint Editor Window for source code display, you need to follow the same steps starting from **Tools -> Preferences -> Report -> General**, and click on the **Default Editor** button.

12. The UDR Violations

After linting the above design, we can also see that there are violations in the **User Defined** rule group. Click on the first violation under rule 30002, you will see the report viewer as follows.

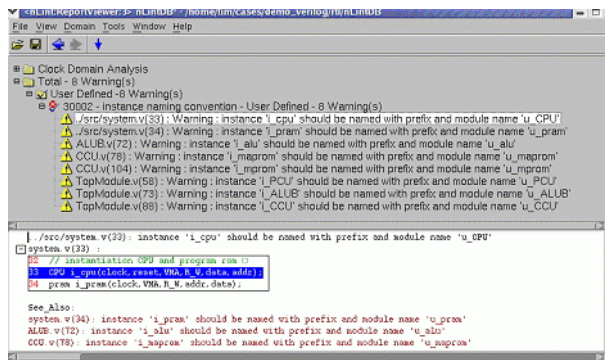


Figure: UDR Violations

Then you can edit or trace source codes with the procedures mentioned earlier.

Clock Source Tree Extraction Tutorial

Overview

In this tutorial, we will guide you through major features of clock analysis with a simple design as an example. The step goes as following:

- Extract clock source tree
- Specify design under analysis
- Load Clock DB into the Verdi system
- Extract mix-clock domain paths
- Specify virtual clock source and primary ports
- Structural proving the mix-clock domain paths with template
- Export information into ASCII file
- Command line options

Prepare the demo cases

Create a working directory and copy the demo files gate.f, top.v and process_gate.v to the working directory. The three demo files are located in <nLint_install_dir>/demo/tutorial/. Please go to the working directory to apply the following steps. Before you begin, please set environment variable as:

```
setenv TURBO_LIBS lsil0k_u
```

And invoke nLint with GUI mode by:

```
nLint -gui -f gate.f
```

Extract Clock Source Tree

After import the design file "gate.f" into nLint by **File -> Import**, invoke command **Run -> Create Domain**.

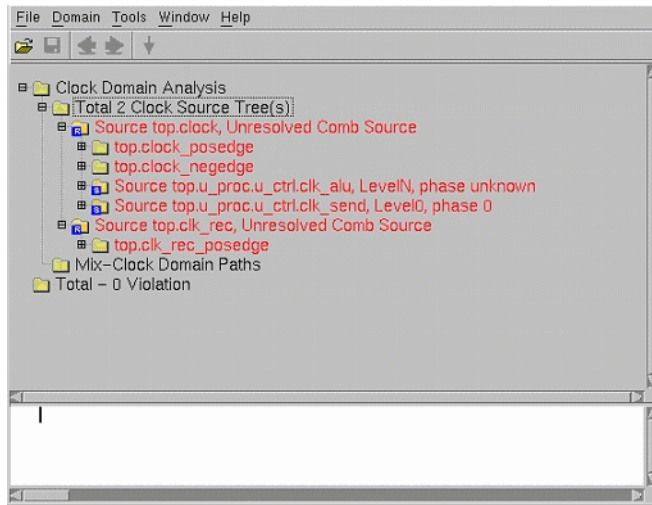


Figure: Result of Extracted Clock Source Tree

In above result, there are two clock source trees. One is driven by `top.clock`, which is un-resolved. The other is `top.clk_rec`, which is also un-resolved.

Invoke the Verdi system with the **Tools -> Debussy** command in the nLint main frame window. Please set the correct path of the Verdi system in **Tools -> Preferences -> Debussy/Verdi**.

The Verdi system will load the same design. Invoke *nSchema* on the top module, you can see there is no port and there are initial statements in it. The clock sources come from the initial statements. So they are all un-resolved.

Specify Design Under Analysis

Most design files include test-bench. It is not convenient to load a separate design files exceptional for nLint. Under such situation, you can set the **Design Under Analysis** in nLint.

Click the **Hierarchy** Tab in nLint main frame window. Use right-mouse-button command on `top.u_proc` to set **Lint From Here** as following figure shows.

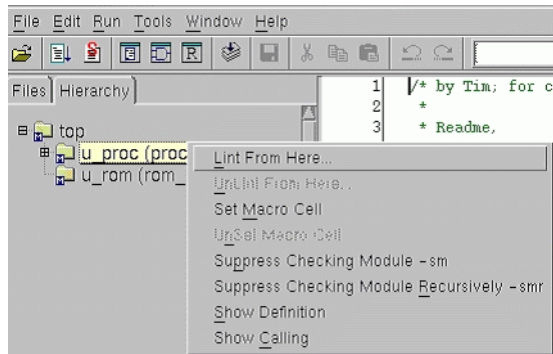


Figure: Invoke Lint from Here

It will invoke a window as following for you to confirm to the setting of **Lint From Here**, or **Design Under Analysis**.

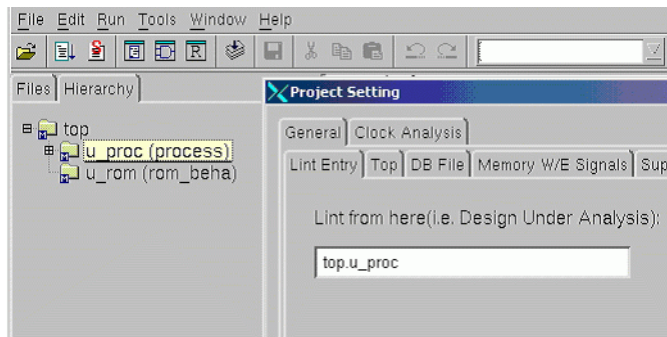


Figure: Design Under Analysis

Click OK to confirm to set the `top.u_proc` as the design for analysis.

By this setting, the boundary of the design will be changed to `top.u_proc`. It means:

- Design out of scope `top.u_proc` will not be analyzed.
- The ports of `top.u_proc` are primary ports of the design.
- But any full scope string will still start from `top`.

Repeat the **Run -> Create Domain** command in nLint main frame window. The result of **Clock Source Tree** extraction is becoming as following figure.

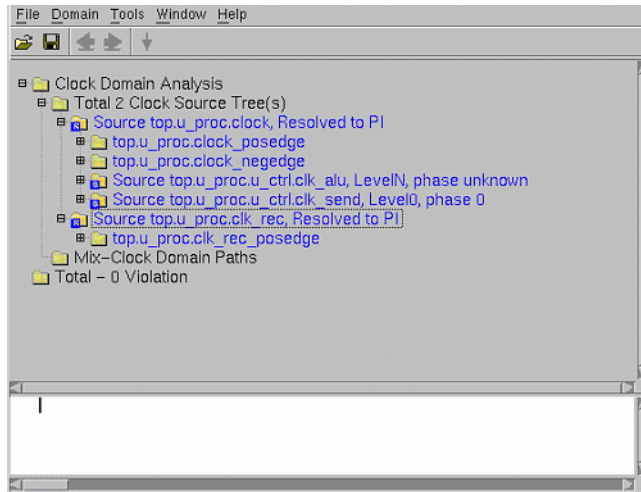


Figure: Clock Source Tree Extraction after Design Under Analysis is set

The two clock sources become resolved to primary input.

Load Clock DB into Verdi

Invoke **Report Viewer** menu command **File -> Save**, to save the clock db. You can load the clock db in the Verdi system by invoking **nSchema -> File -> Load Clock Domain Results**. Choose the clock db saved from nLint to load. Then the clock source tree will be listed in the Verdi system as following.

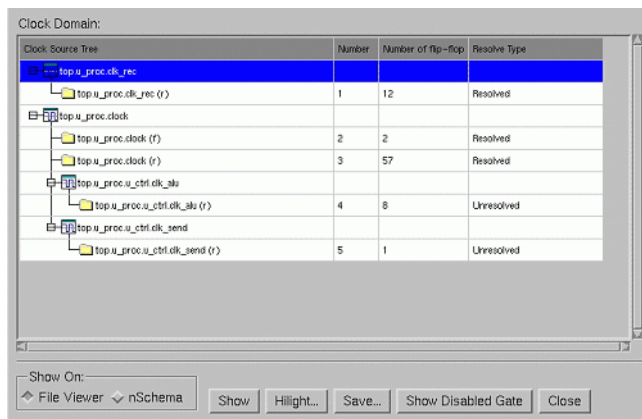


Figure: Clock Source Tree in Verdi

Then you can put the clock source tree, internal clock source or the clock domain interested into *nSchema*. Following is *nSchema* view of clock source `top.u_proc.clock`.

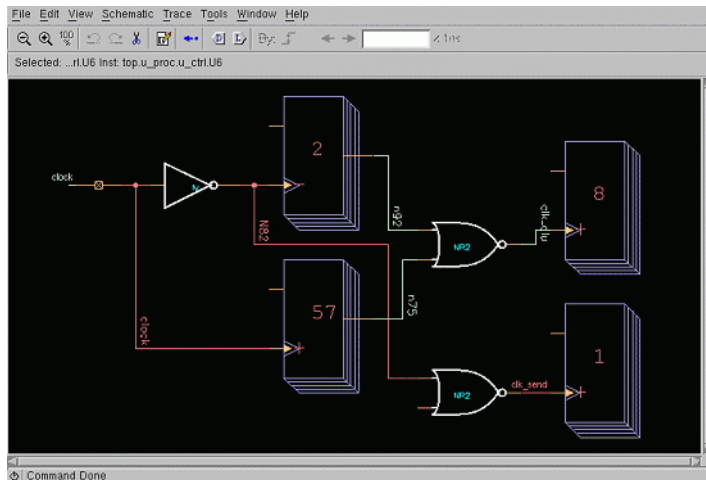


Figure: *nSchema* view of clock source tree `top.u_proc.clock`

Extract mix-clock domain paths

Then we can continue to analyze the crossing paths between clock domains. Invoke **Report Viewer** menu command **Domain -> Mix Clock Domain Extraction**. A window as following will be popped up.

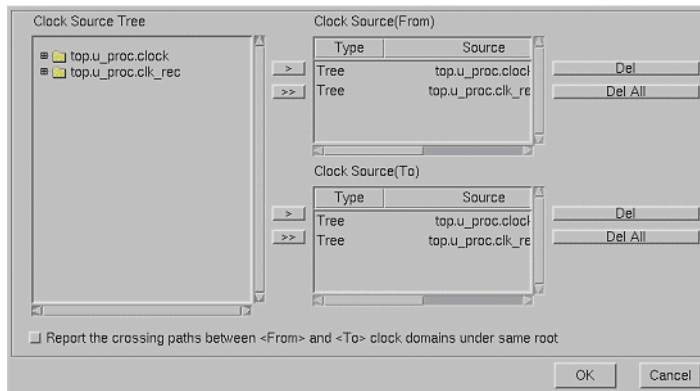


Figure: Set Mix Domain

Add the two clock source trees to both of the **Clock Source (From)** and **Clock Source (To)** boxes. It directs nLint to check the crossing path from clock source

tree `top.u_proc.clock` to clock source tree `top.u_proc.clk_rec` and vice versa.

Click OK to extract the crossing path. The result is shown in following.

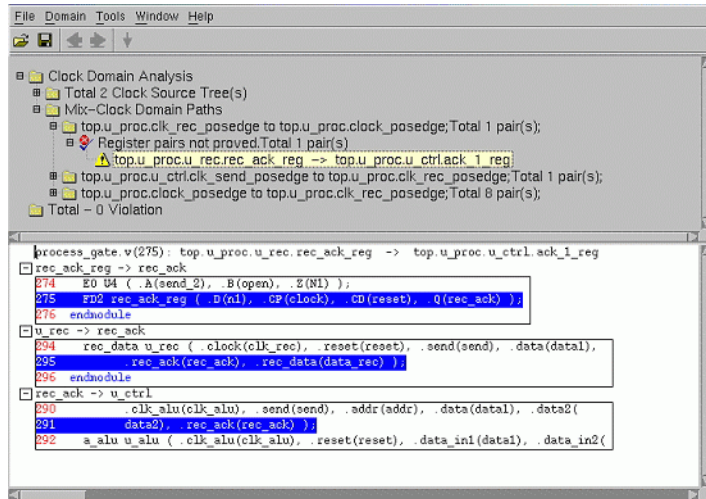


Figure: The result of Crossing Path

In the result, the crossing paths are grouped by domain pairs. From beginning to the end, there is one crossing path from domain positive triggered by `top.u_proc.clk_rec` to the domain positive triggered by `top.u_proc.clock`; there is one crossing path from domain positive triggered by `top.u_proc.u_ctrl.clk_send` to domain positive triggered by `top.u_proc.clk_rec`; there are 8 crossing paths from domain positive triggered by `top.u_proc.clock` to domain positive triggered by `top.u_proc.clk_rec`.

If you expand the folder of the domain pair, you can find the registers information of the crossing path. Also in detail window, the path information is listed from the start register to the end register.

If double clicking the register pair, nLint will show the path of register pair in *nSchema*. Following is the result if you double click the highlighted register pair in above figure.

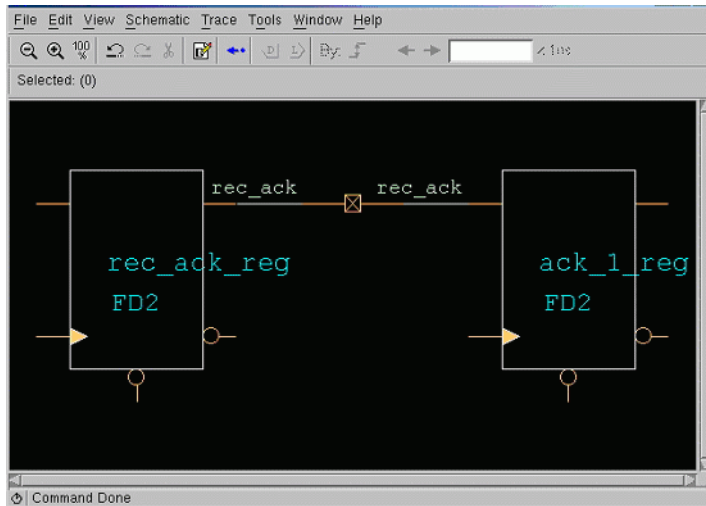


Figure: Crossing Path shown in nSchema

Specify virtual clock source and primary ports

In above crossing path extraction, we do not consider the primary ports. For example, the primary input port `top.u_proc.cmd_data` comes from outside block and sampled by another clock `clock_ram`. We can consider such primary ports as sequential elements in extracting crossing path.

In nLint main frame window, invoke **Run -> Project Setting -> Clock Analysis -> Virtual Clock Source**, you can specify a virtual clock named `clock_ram` as following.

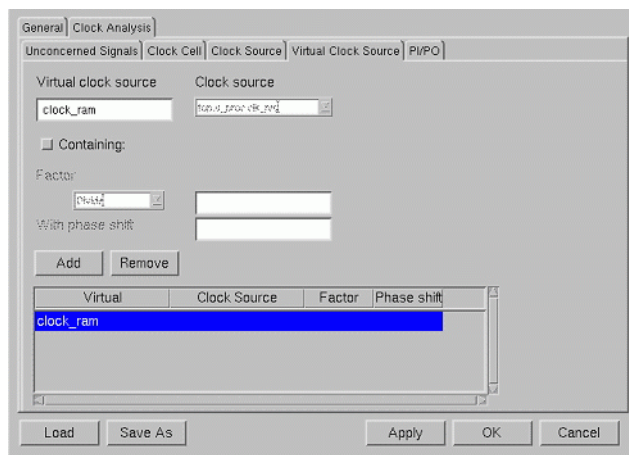


Figure: Specify virtual clock source `clock_ram`

Click **Apply** to enable the setting.

After that, you can specify the primary ports as virtual clock source. Choose the `clock_ram` in Clock Source field. Select **pos-edge** box as the trigger edge. Highlight the primary port `top.u_proc.cmd_data` and apply the clock source on it.

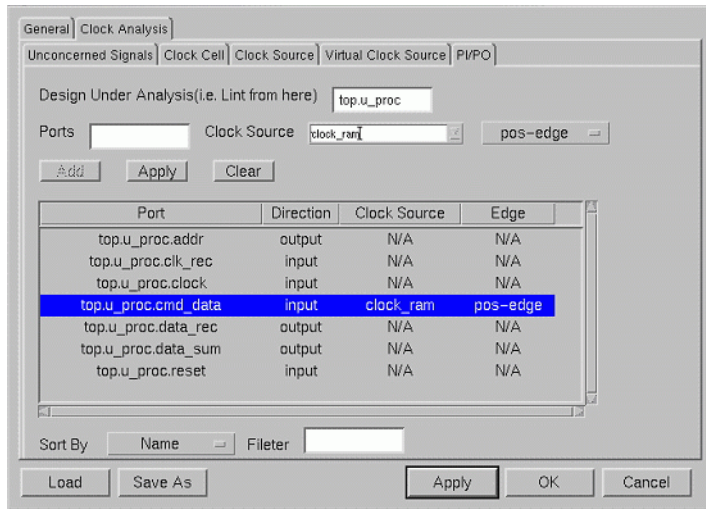


Figure: Specify primary port for virtual clock source

Click **Apply** to let the setting take effect.

After that, repeat the clock source tree extraction by **Run -> Create Domain**. The clock source tree with virtual clock source `clock_ram` as root will be listed as following.

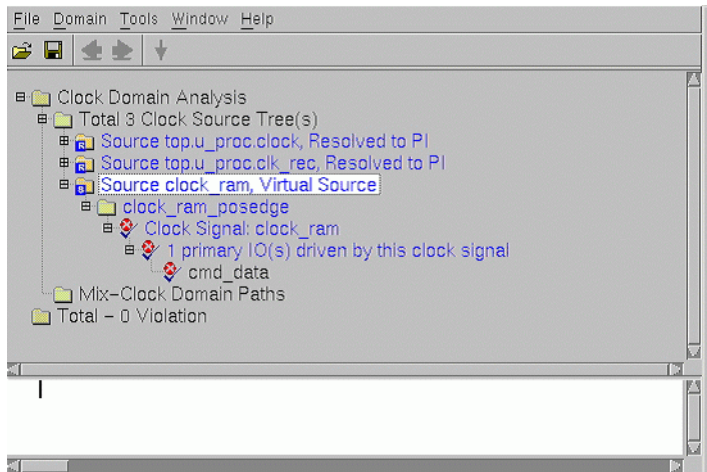


Figure: Virtual Clock Source Tree

The primary port `cmd_data` is listed under the virtual clock source tree like registers.

Repeat the crossing path extraction command **Domain -> Mix Clock Domain Extract**. And add the virtual clock source tree into both the **From Domain** and **To Domain** lists.

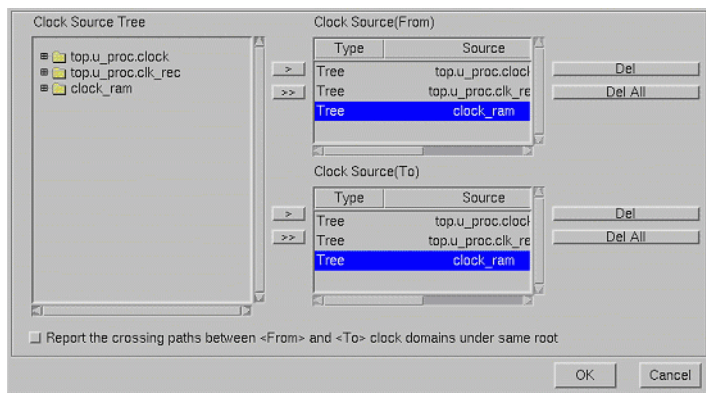


Figure: Add Virtual Clock Source Tree in Mix-Clock Domain Extraction

After click OK, additional crossing path from the primary input port `cmd_data` will be extracted as following.

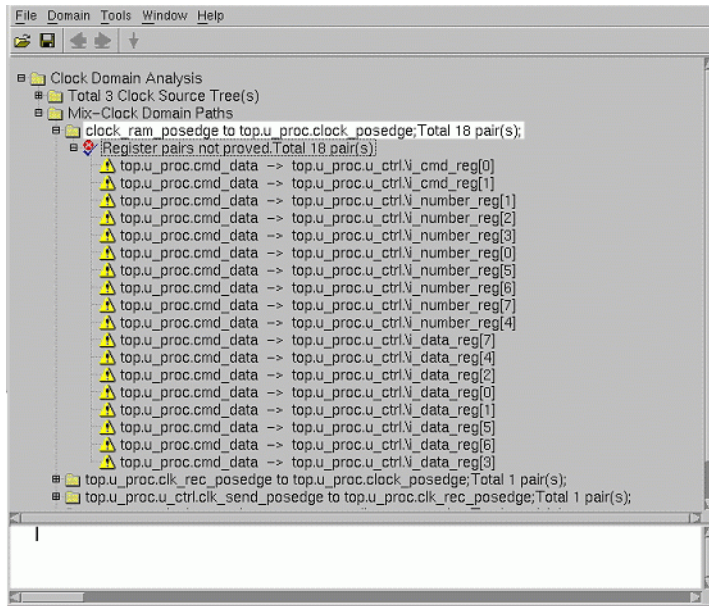


Figure: Crossing Path from Primary Input Port

There are 18 crossing paths from primary input port `cmd_data` to registers of domain positive triggered by `top.u_proc.clock`.

Structural proving the mix-clock domain paths with template

Generally, there is design rule to force crossing path pass through a number of synchronizer flip-flops. In nLint structure synchronizer template is provided to prove the design rule.

Invoke the command **Domain -> Synchronizer Proving** in Report Viewer, the following box will be popped up.

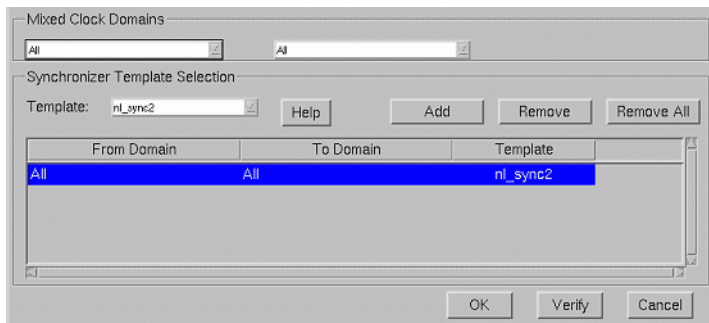


Figure: Synchronizer Proving Box

Choose proving crossing path from "All" clock domains to "All" clock domains. It means all crossing paths found will be applied on such proving. Choose template n1_sync2. It is a default template. Click **Help** button will show the detail information about the template as following.

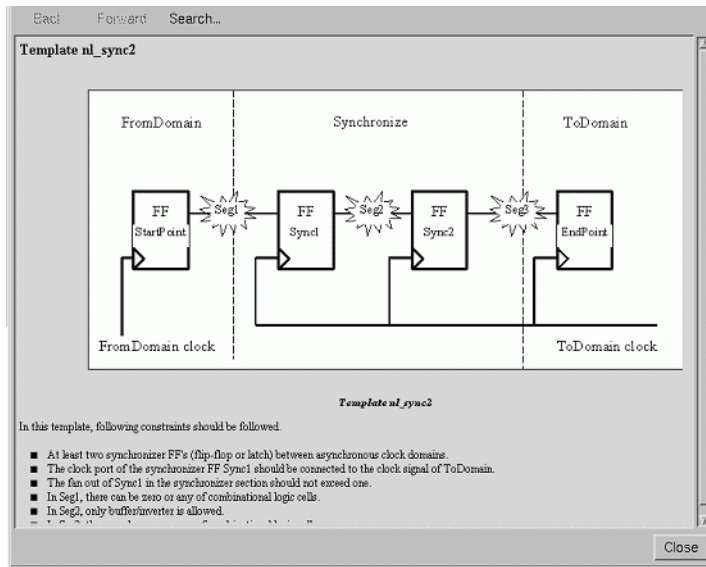


Figure: Help Information for template n1_sync2

Click **Verify** to start the proving process. The result is shown as following.

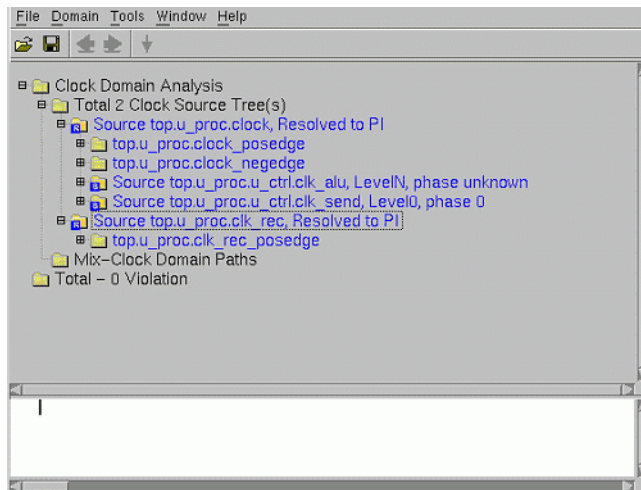


Figure: Synchronizer Proving Result

For satisfied path, for example, the path from `top.u_proc.u_ctrl.send_reg` to `top.u_proc.u_rec.send_1_reg`, which is expanded in above figure, will be expanded with the synchronizer flip-flops. Double click the expanded path, it will be shown into *nSchema* as following.

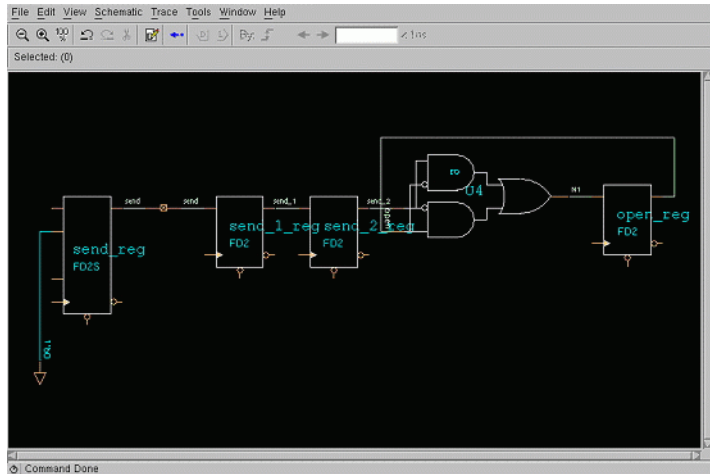


Figure: Satisfied Crossing Path

From *nSchema*, it is clear that the signal `send` passing through 2 synchronizer flip-flops before goes into destination clock domain.

Another expanded path is an unsatisfied path, from `top.u_proc.u_ctrl.\data_reg[0]` to `top.u_proc.u_rec.\rec_data_reg[0]`. The failure reasons are printed out as a message under the path. It says "the cross path does not have enough number of FFs, the least is 2". It means the signal does not pass through 2 synchronizer flip-flops before goes into destination clock domain.

Export information into ASCII file

All above result shown in **Report Viewer** could be exported into an ASCII file. Invoke Report Viewer menu command **File -> Export Clock Domain**, a form as following will be prompted.

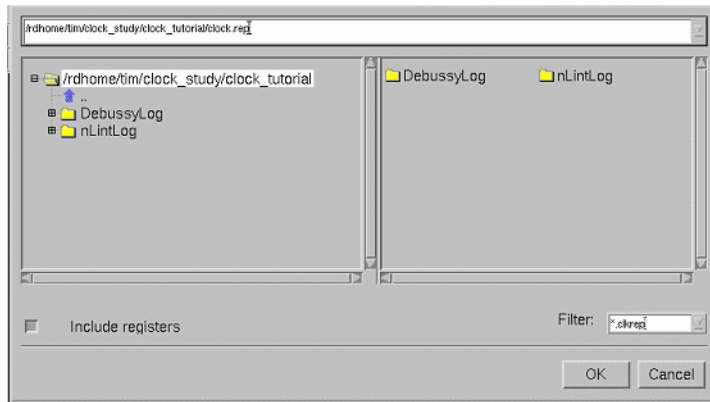


Figure: Export Clock Domains

Choose to export all registers information under clock source tree by check the checked box **Include registers**. Click OK to export the information. The result is in clock.rep.

Command line options

All above actions can be run quickly by applying the following command line options.

```
-lintTop top.u_proc
-vclk_source clock_ram
-io_clk top.u_proc.cmd_data clock_ram_posedge
-ex_clk
-clk_from_domains All
-clk_to_domains All
-sync_p All:All:nl_sync2
-clk_export clock.rep
-clk_export_reg
```

You can write it into a file clock.opt and use following command to invoke nLint.

```
nLint -gui -f gate.f -opt clock.opt
```

Then you can get the final result directly.

User Interface

Overview

nLint main frame window in graphical user interface (GUI mode) is invoked by the `nLint -gui` command. This nLint main frame window is comprised of three window panels:

- Project Window, at the upper left part of the main frame window, displays source files of imported design and design hierarchy in Files and Hierarchy tabs respectively.
- Editor Window, at the upper right part, shows source code of the selected file in the Project Window.
- Output Window, at the lower half part, serves as a *console* of the program. Console messages printed out by nLint during checking are collected and displayed in this window.

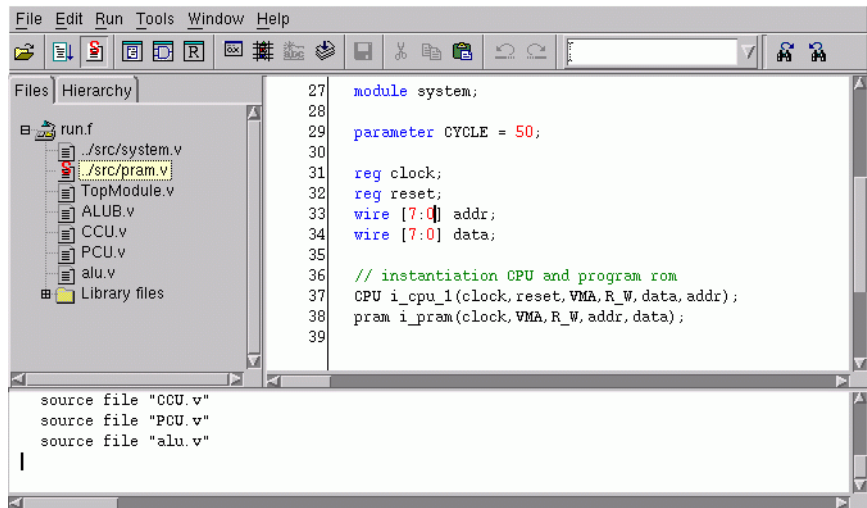


Figure: nLint Main Frame Window


Files Tab

After you have imported a design, a list of source files will be shown in a tree structure. Double-clicking on a file in the tab displays the file contents in the

integrated Editor Window, at the upper right part. This is a quick way to see the contents of a file.

Under the **Files** tab, clicking the right mouse button on a file appears a shortcut menu, which allows you to open the file in another Editor Window and toggle the file status to be checked or unchecked.

On a node of file in the design file tree, clicking the right mouse button leads to a shortcut menu, which allows you to do the following operations:

- **Open:** this command brings up a separated Editor Window for file editing if the file is not shown in the Editor Window of nLint main frame window.
- **Suppress File -df:** this command toggles the suppress action to the violations in design file. The suppressed node is represented by a  icon. The "-df" means the behavior is the same as the command line option -df.

Hierarchy Tab

Initially, the source files of imported design are loaded in the Files tab. When you click on the Hierarchy tab at the first time after loading the design files, nLint will compile your data and show the information of design hierarchy on this tab, as shown below. If your design is imported from library, nLint will directly show the design hierarchy.

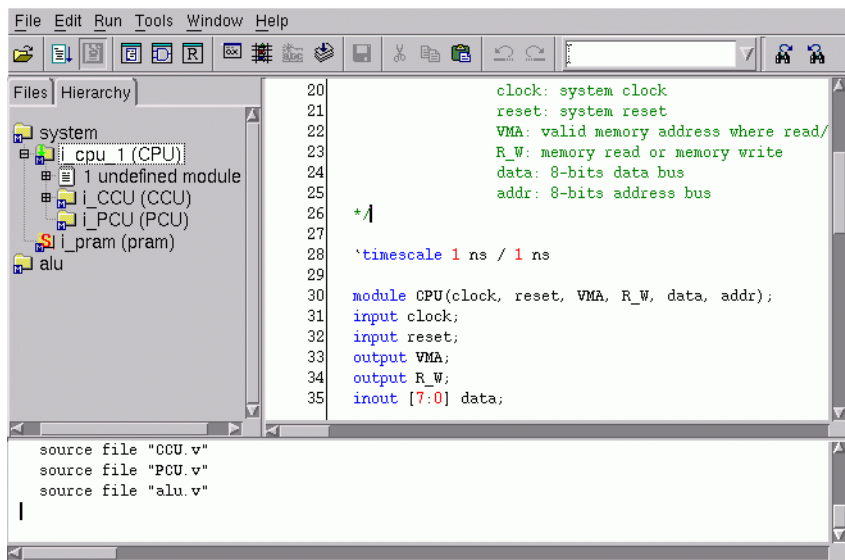





Figure: Hierarchy Tab under Project Window

On a node of instance (module) in the design hierarchy tree, clicking the right mouse button appears a shortcut menu, which allows you to do the following operations:

- **Lint from here:** this command jumps to the **Lint entry** tab of **Project Setting** form with the string of current scope appears at the **Lint from here** text box. The node with **Lint from here** is displayed as a  icon.
- **UnLint from here:** this command removes the setting of **Lint from here**. It is available when the node is set as "**Lint from here**".
- **Set Macro Cell:** this command sets the module as **Macro Cell** (or so called black box). The node of **Macro Cell** is displayed as a  icon. It is available when there is no other setting on the node. The related command line option is -bb.
- **UnSet Macro Cell:** this command removes the setting of **Macro Cell**. It is available when the node is set as **Macro Cell**.
- **Suppress Checking Module -sm:** this command toggles the suppress action to the violations in module. The suppressed node is displayed as a  icon.
- **Suppress Checking Module Recursively -smr:** this command recursively toggles the **Suppress Checking Module** for all the modules under this node. When it is checked, nLint will go through all the nodes under this starting node to set **Suppress Checking Module**. The behavior is similar as going through all the modules under the node recursively and manually set the **Suppress Checking Module**. When it is unchecked, nLint will go through all the nodes under the starting node to remove setting **Suppress Checking Module**. The behavior is similar as going through all the modules under the node recursively and manually removing the setting **Suppress Checking Module**.
- **Show Definition:** this command shows the source code at the module definition of the node in Editor Window.
- **Show Calling:** this command shows the source code at the instance of the node in Editor Window.

The menu commands in nLint main frame window are described in the following sections.

File Commands

Import Design

Toolbar Button:



This command opens the *Import Design* form where design files to be imported can be specified.

NOTE: When a VHDL design is loaded, a *novas.rc* file is needed for mapping the library.

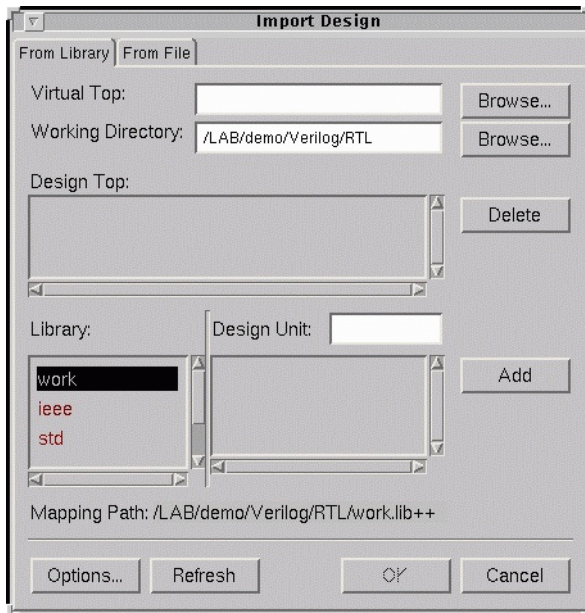


Figure: Import Design Form - From Library

The *Import Design* form includes the **From Library** and **From File** tabs:

From Library Tab

This command imports a Verilog or VHDL design from a pre-compiled library. The Library browser in the **From Library** tab lists all top objects from the selected pre-compiled library.

Select a design with the browser or the command line window then click **OK**. You can incrementally compile Verilog and VHDL source into the defined libraries.

Either importing a Verilog or a VHDL library, you can select multiple top modules in the Design Unit list or specify the top modules in the text box. Specify the working directory of the library in the Working Directory text box. Click **Refresh**, if you want to see the updated library generated in the working directory.

NOTE: If you import a library from **Virtual Top**, **Design Top** will be disabled.

Clicking the **Options** button displays the *Library Options* form. You can specify the library name, library path and set the incremental load option in the dialog box.

From File Tab

This tab allows you to open the design and pre-compile the source code into nLint's proprietary data format. You can also specify Verilog/VHDL compilation options in addition to file names.

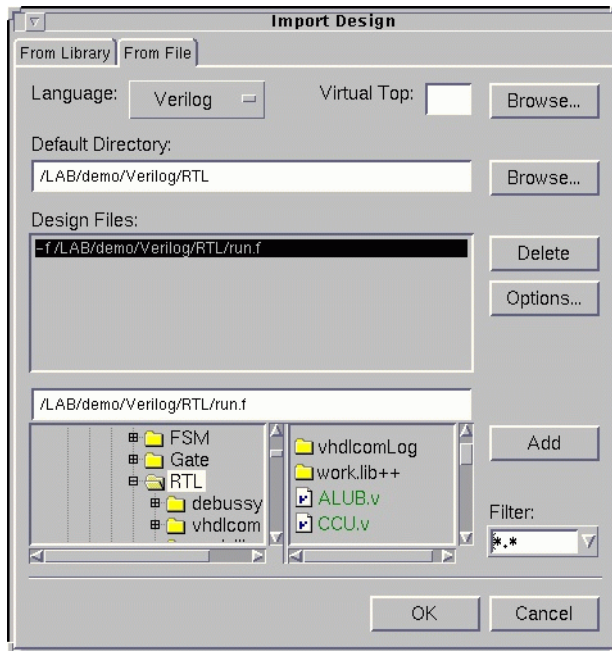


Figure: Import Design From File Form

You can select the design language to be imported in the Language options: Verilog, Verilog-2001, SystemVerilog, VHDL-87, VHDL-93 or VHDL-2000. For VHDL Language, a *Set Top Module* form opens and asks you to select the top design unit when you are ready to import the VHDL design if it is your first time to import the VHDL design. According to VHDL definition, to elaborating a design, a top design unit specified as the elaboration start point is necessary. When nLint compiles (it may be invoked by clicking the **Hierarchy** Tab or **Lint** command) all design files, if the top design unit is not specified, a dialog as following will be popped up to prompt you a top module.

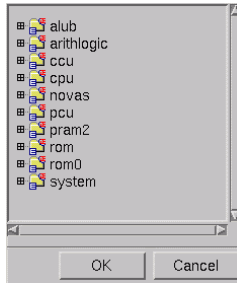


Figure: Set Top Module Form

You can specify the **Default Directory** by either typing the path directly in the text box or clicking **Browse**. Or, when you add any *run.f* to the Design Files list, the directory path for this *run.f* file is automatically updated to the text box.

When you click the **Options** button, the *Import Design Options* form will open.

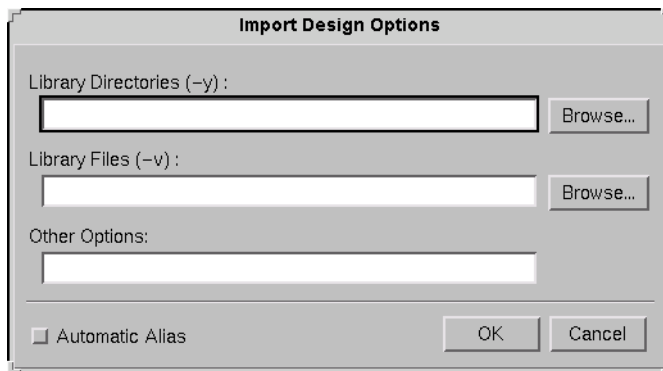


Figure: Import Design Options Form

You can specify the **Library Directories**, **Library Files** or **Other Options** for your current design. These options serve as a command line option when you import a design file into nLint. Note that if you have already specified your

options using the **Importing Design** page under the **Tools -> Preferences -> Source Code** page (which is the global setting), these options will be copied into the *Import Design Options* form. You can specify different options in the *Import Design Options* form, but these options are only valid for the current design.

Library Directories (-y): Specifies the library directory, which appends to the command line option `-y`. Type in the directory path or click the **Browse** button to browse using the *Choose Library Directory* form.

Library Files (-v): Specifies the library file, which appends to the command line option `-v`. Type in the file name or click the **Browse** button to browse using the *Choose Library File* form.

Other Options: See the supported Verilog reference manual for available options. For these options, you can invoke the Verdi system with the command option `-h`.

Automatic Alias: This option instructs the Verdi system to automatically create signal aliases according to Verilog's constant parameter. If you create a parameter as a constant and assign or compare signals with that parameter, the Verdi system will automatically create the a signal alias as that parameter.

Save File

This command saves the modified data in Editor Window and closes the window. It is only available when the file is modified. If you edited the source code and didn't save the changes before leaving the Editor Window, nLint will pop up a Question dialog box, as shown below, to ask if you wish to save the changes or leave the source code file unchanged.

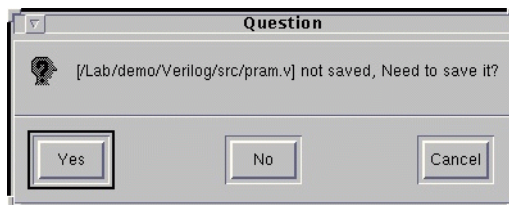


Figure: Question Dialog Box

Save As

This command allows you to save the current file in an integrated Editor Window with different file name via the *Save As* form.

Export Options

This command allows you to export the current setting of nLint as command line options and save them into a file. The file extension is **.opt*.

When running nLint under batch mode, you can easily apply the current setting of nLint by using `-opt <option_file>`, especially those options such as

- Partition setting
- Clock domain setting

Restore Session

This command brings back your previous working environment, which includes such information as the window location, sizing, the opened design, etc. This command also allows you to continue your previous checking session without having to go through the set-up steps repeatedly. The default file extension is **.ntp*.

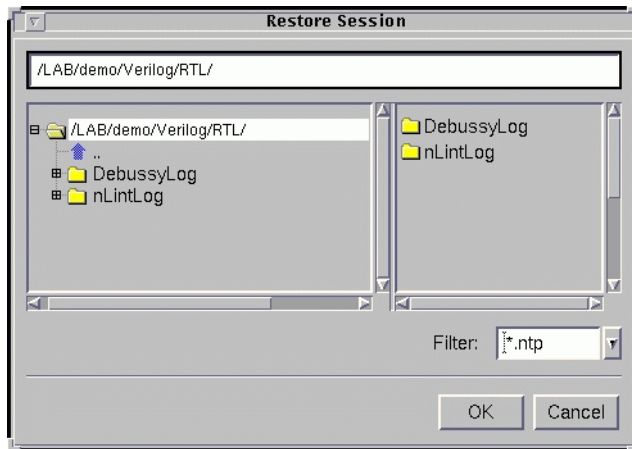


Figure: Restore Session Form

Save Session

This command allows you to save the current checking session to a given session name via the *Save Session* form, so that you can restore it later. By default, the file will be saved with the **.ntp* file extension. The session files are shown automatically when you invoke **Restore Session**.

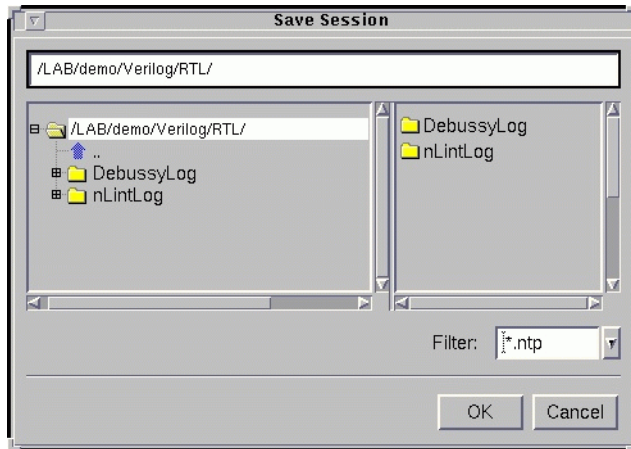


Figure: Save Session Form

Load Report DB

You can invoke this command to open a report data base. After you specify a report data base directory, *Report Viewer* will load the checking information from the report data base and show the information in the tree window of *Report Viewer*.

Save Report DB

You can invoke this command to save the modified report data base files.

Save Report DB as

You can invoke this command to save the report data base files with another file name. You can save obsolete violations in violation data base by enabling the **Save Obsolete Violations** option. This option is *on* by default.

Suppress

Toolbar Button: 

When you are working under the **Files** tab of Project Window, this command toggles the check or uncheck status for the selected source file. When you are

under the **Hierarchy** tab, this command toggles the suppress action to violations for the selected module.

On the other hand, clicking the right mouse button on the selected source file or module also allows you to operate this command from a shortcut menu.

File Order

Options below are available when you are working under the **Files** tab.

Original Order

This command keeps the files shown in the Project Window in their original order when they are imported.

Sort File By Ascending

This command specifies that files shown in the Project Window are sorted in ascending order (A to Z or zero to 9).

Sort File By Descending

This command specifies that files shown in the Project Window are sorted in descending order (Z to A or 9 to zero).

Exit

This option closes all windows displayed in nLint with confirmation message if there is any modification in your design.

Edit Commands

Undo

Toolbar Button: 

Bind Key: CTRL+Z

This command reverses the last action you performed. Notice that if you later decide you didn't want to undo an action, you can invoke the **Edit -> Redo** command.

Redo

Toolbar Button: 

Bind Key: CTRL+Y

This command repeats the last action you performed.

Cut

Toolbar Button: 

Bind Key: CTRL+X

This command removes the selected section and places a copy on the clipboard.

Copy

Toolbar Button: 

Bind Key: CTRL+C

This command places a copy of the current selection on the clipboard.

Paste

Toolbar Button:



Bind Key:

CTRL+V

This command places the information, which you cut or copy to the clipboard, into the Editor Window at the cursor position.

Delete

Bind Key:

Delete

This command removes the selected text. Note that the deleted text will not be copied to the clipboard.

Select All

This command selects the contents in the Editor Window.

Find

Toolbar Button:



Bind Key:

CTRL+F

This command helps you locate the text that you wish to find within the source lines via the Find text box. The **Match Case** and **Regular Expression** options are used for text searching.

With the **Match Case** option, the search will only look for those instances in which the capitalization matches the text you typed in the Find text box.

With the following regular expressions, you can fast locate the text you need:

Expression	Description
[1]	char Matches itself, unless it is a special character (meta-char): . \ [] * + ? ^ \$
[2]	. Matches any character.

Find Next

Toolbar Button: 

Bind Key: F3

This command finds the next instance that you specified in the Find text box.

Replace

Bind Key: CTRL+H

This command finds and replaces the text you specified via the *Replace* form.

Advanced

Indent

Bind Key: Tab

This command indents the current line or selection. To do so, highlight the selection before you invoke this command.

Outdent

Bind Key: SHIFT+Tab

This command outdents the current line or selection. To do so, highlight the selection before you invoke this command.

Comment

This command adds a line comment symbol for each selected source line. To do so, highlight the selected line before you invoke the command. The text color will be changed (from black to green) after adding the line comment symbol.

Uncomment

This command deletes the first line comment symbol for each selected source line. To do so, highlight the selected line before you invoke the command. The text color will be changed (from green to black) after deleting.

Match Brace

Bind Key: CTRL+<bracket right>

This command is used to locate the mate for a certain delimiter. Place the cursor on a brace (parenthesis, brace, and bracket) and select the **Match Brace** command. The text from the selected brace to its matching brace will be highlighted.

Bookmarks

Toggle Bookmark

Bind Key: CTRL+F2

Use this command to set or clear the bookmark at the current line of the source code.

Next Bookmark

Bind Key: F2

This command jumps to the next bookmark.

Previous Bookmark

Bind Key: SHIFT+F2

This command goes back to the previous bookmark.

Clear All Bookmarks

This command clears all bookmarks you set in the Editor Window.

Run Commands

Project Setting

This command allows you to set relevant information for the current project, for example, specifying the lint entry, the top module, the report DB file and the memory write enable signals and the clock source, etc.

The following common buttons appear at the bottom of each tab.

- The **Load** button allows you to load a project setting file and apply the specification.
- The **Save As** button allows you to save the current project setting into a file.
- The **Apply** button allows you to execute current changes of setting.
- The **OK** button allows you to execute current changes of setting and close the form.
- The **Cancel** button allows you to discard the current modification and close the form.

General Tab

Lint Entry

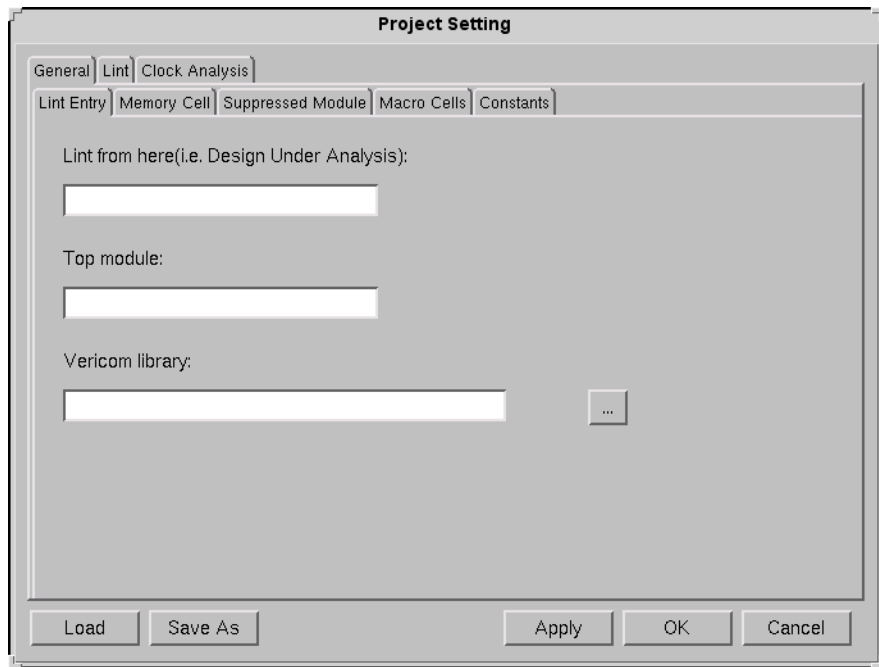


Figure: Project Setting -> General -> Lint Entry Sub-tab

In the **Lint from here (i.e. Design under Analysis)** text field, you can specify a full hierarchy scope and skip linting the testbench. nLint will treat the specified scope as a sub design.

The setting also impacts to clock analysis. The specified entry scope is treated as design under analysis. You can exclude the test-bench from analysis without changing the import files.

In the **Top Module** text field, you can specify the top module for the current project. It is necessary for VHDL design. If it is not specified, nLint will show a **Set Top Module** window and ask you to choose the top module when elaborating the design.

In the **Vericom library** text field, you can use it to export the vericom library, so the directory will be valid to vericom and writable.

You can specify a vericom library directory in the **Vericom Library** option Before you invoke the **Run -> Local Lint** command, so that nLint will compile design incrementally and save the result to the specified library.

Design Entry Sub-tab

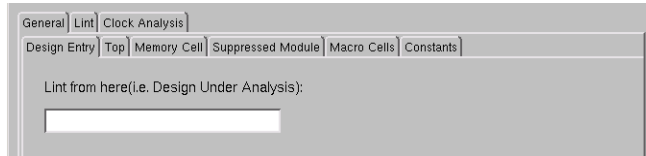


Figure: Project Setting -> General -> Lint Entry Sub-tab

The **Design Entry** sub-tab allows you to specify a lint process from the sub-design. By default, nLint will lint from the top module. This is different with the setting in the **Top** sub-tab. nLint will only lint the sub-design and elaborate the whole design when a sub-design is specified in the **Design Entry** sub-tab.

The setting also impacts to clock analysis. The specified entry scope is treated as design under analysis. You can exclude the test-bench from analysis without changing the import files.

Top Sub-tab

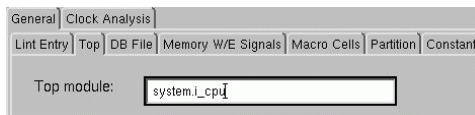


Figure: Project Setting -> General -> Top Sub-tab

The **Top** sub-tab allows you to specify the top module for the current project. Specifically, it is necessary for VHDL design. If it is not specified, nLint will prompt a **Set Top Module** dialogue box and ask you to choose the top module when elaborating the design.

For Verilog design, it is used only when there is more than one top module in the design files.

Memory Cell Sub-tab

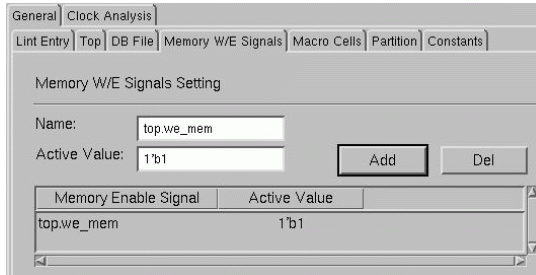


Figure: Project Setting -> General -> Memory W/E Signals Sub-tab

The **Memory Cells** sub-tab allows you to specify the memory write enable signals with its active value. This is used in checking rule 22201. For details on this rule, refer to [Rule Category](#).

You may specify a signal name, which should be a full hierarchy name, in the **Name** text box; and specify a value for the active signal in the **Active Value** text box.

The **Add** button allows you to add a signal with value into the signal list.

The **Del** button allows you to delete a selected item from the signal list.

Suppressed Module Sub-tab

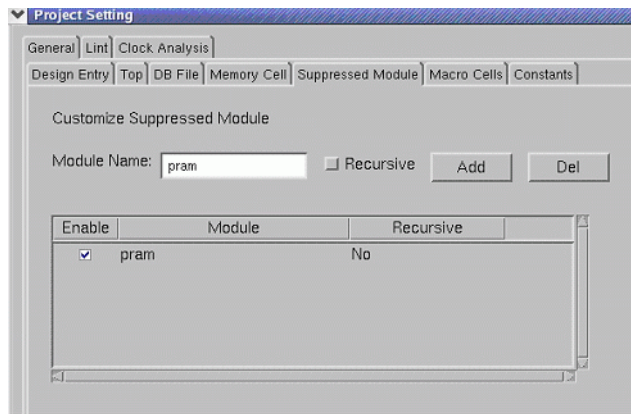


Figure: Project Setting -> General -> Suppressed Module

The **Suppressed Module** sub-tab allows you to specify a module to be suppressed. If a module is suppressed, all violations detected in the module will be suppressed. If a module is suppressed recursively, all the modules traversed from the module specified will be suppressed.

The **Add** button allows you to add a module name into the module name list.
The **Del** button allows you to delete a selected item from the module name list.
The **Recursive** check box allows you to specify if the module is suppressed recursively or not.

Macro Cells Sub-tab

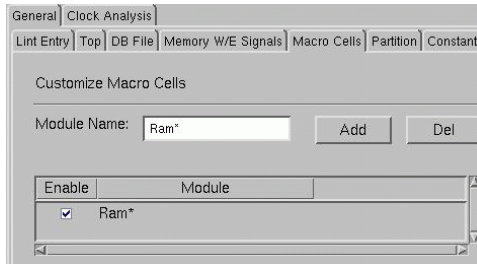
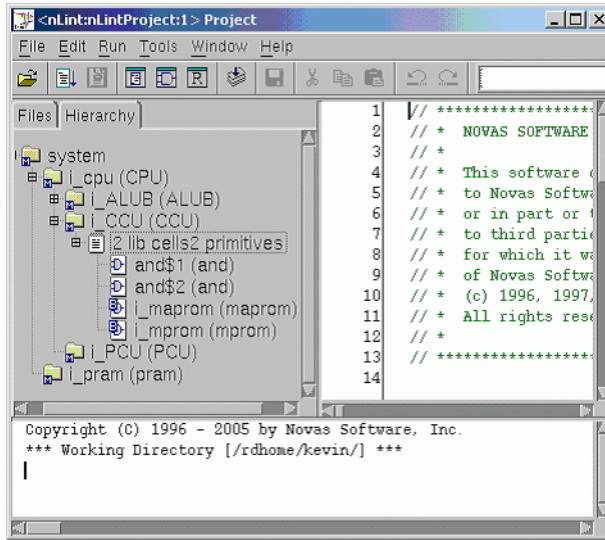


Figure: Project Setting -> General -> Macro Cells Sub-tab

The **Macro Cells** sub-tab allows you to specify the module as a black box. If a module is defined as a macro cell, nLint will not traverse into the module except that port connection for upper level. Thus all the design under the module will not be checked by nLint. The command line option is `-bb`.

Please note that primitive can't be set as macro cell. Whether the specified macro cell name includes a wild card or not, nLint will give a warning message if the specified macro cell name is not valid. After the macro cell setting is applied, in the hierarchy tab of the main frame window, an icon with a capitalized B will be displayed before the module. See the figure below for an example.



The **Module Name** text box is used to specify the module name to be treated as macro cell.

The **Add** button allows you to add a module name into the module name list.

The **Del** button allows you to delete a selected item from the module name list.

The checkbox in front of each row allows you to enable or disable the usability of the row.

With the checkbox **Expand wildcard when save**, you can choose to save the original wildcard specification into the option file or to choose to save the matched module names into the option file.

Constants Sub-tab

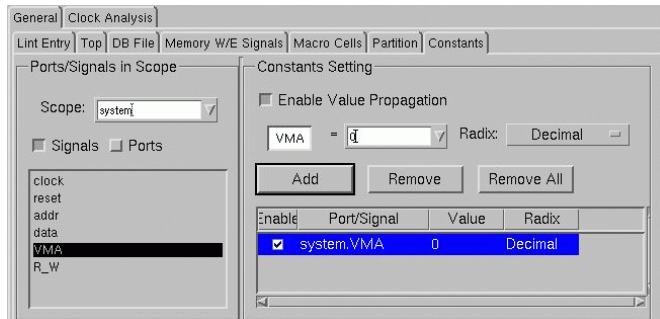


Figure: Project Setting -> General -> Constants Sub-tab

This **Constants** sub-tab specifies constant on the primary input port or internal port/signals. When you lint a design with scan inserted logic, you may want to ignore those tool-inserted logics. Under this case, you can set the scan mode to an inactive value. nLint will propagate the constant value as much as possible in the design. After that, the uninterested logic will be reduced. All DFT violations like *gated clock* will be affected by this setting.

Under **Ports/Signals in Scope**, the Scope combo box allows you to enter the interested scope or to select a scope in the history list. The **Signals** and **Ports** check boxes allow you to switch to show signals or ports under the scope.

Under Constants setting, value propagation is performed when the **Enable Value Propagation** checkbox is checked. The left text box allows you enter a port/signal with full hierarchy name; or you can choose a port/signal from the list box under Ports/Signals in Scope. The text box, right to the equal symbol, allows you to enter the constant you want to apply on the signal. The Radix combo box allows you to select the radix of the constant value.

To add a specified constant with port/signal name and value in the list window, use the **Add** button. The **Remove** button deletes a selected row from the list window while the **Remove All** button deletes all.

The checkbox in front of each row allows you to enable or disable the usability of the row.

When propagating the constant, nLint will try to propagate the constant value as fast as possible. This means that if the output value of the instance is also a constant, when applying the specified constant, nLint will continue to propagate the output value only if the function of the instance is known.

If using symbol library, nLint can only propagate the constant through the following symbols:

```
AND NAND OR NOR XNR XOR BUF INV BUFD BUFIF0 BUFIF1 NOTIF0 NOTIF1  
DFE MUX
```

or the symbols with equation when generating the symbol library.

Lint Tab

Partition Sub-tab

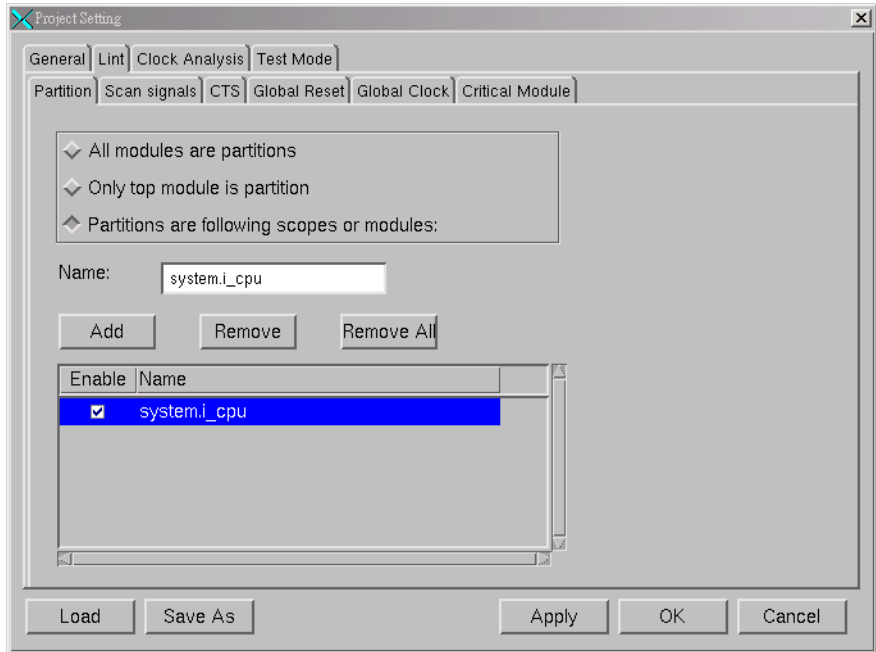


Figure: Project Setting -> Lint -> Partition Sub-tab

This **Partition** sub-tab allows you to set partition of the design. The setting will be used in checking some rules, such as:

- 25015, outputs leaving partition without been driven by register.
- 25011, input with heavy transitive fan out of end points.
- 25013, output with heavy transitive fan in of start points.

You can check any of the following check boxes at the same time.

All modules are partitions: when it is checked, all modules are treated as individual partitions. For example, rule 25015 will check the output port on each module.

Only top module is partition: when it is checked, there is only one partition in the design, that is, the top module. For example, rule 25015 will check the output port on top module only.

Partitions are the instances under following scopes: it is used to create partitions on instances under specified scopes. For example, in the **Partition** dialog box above, the instances under scope *system* are all partitions.

The **Add** button allows you to add a blank item. After a blank item is inserted, click on it and enter the scope you want. The **Enable** field in the list box allows you to enable/disable the item. The **Remove** button allows you to remove a selected item from the list box while the **Remove All** button removes all selected items.

The checkbox in front of each row allows you to enable or disable the usability of the row.

Scan Signals Sub-tab

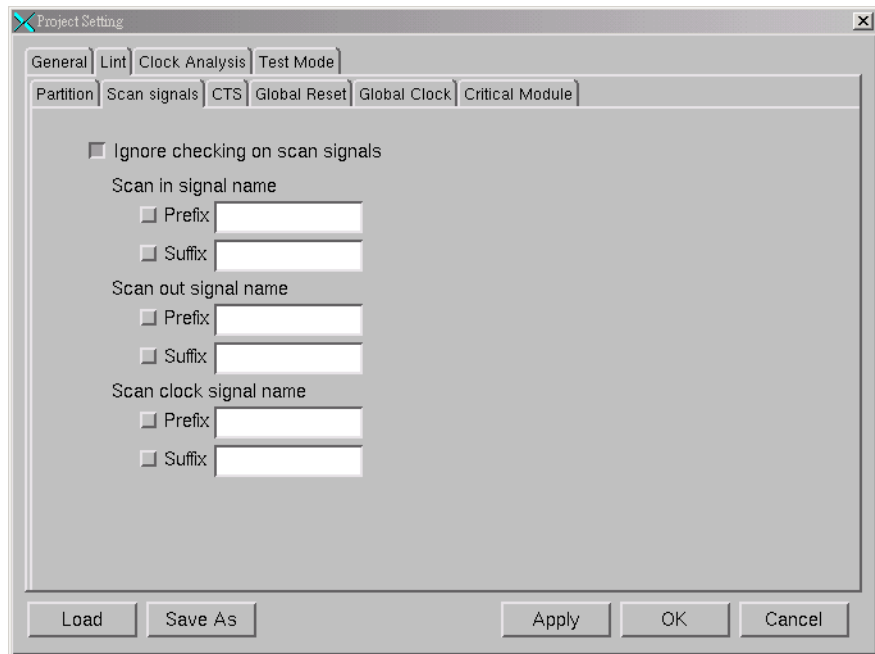


Figure: Project Setting -> Lint -> Scan Signals Sub-tab

In the Scan Signals sub-tab, the settings are described as follows:

- **Ignore checking on scan signals:** when it is checked, the scan signals will be ignored in all connection-related rules checking.
- **Scan in signal name prefix/suffix:** to specify the prefix/suffix of scan in signal name. If a signal's name satisfies this condition, it is treated as scan in signal.

- **Scan out signal name prefix/suffix:** to specify the prefix/suffix of scan out signal name. If a signal's name satisfies this condition, it is treated as scan out signal.
- **Scan clock signal name prefix/suffix:** specify the prefix/suffix of scan clock signal name. If a signal name satisfies this condition, it is treated as scan clock signal.

In all the fields for prefix and suffix, you can use comma to separate more than one string for prefix or suffix.

CTS Sub-tab

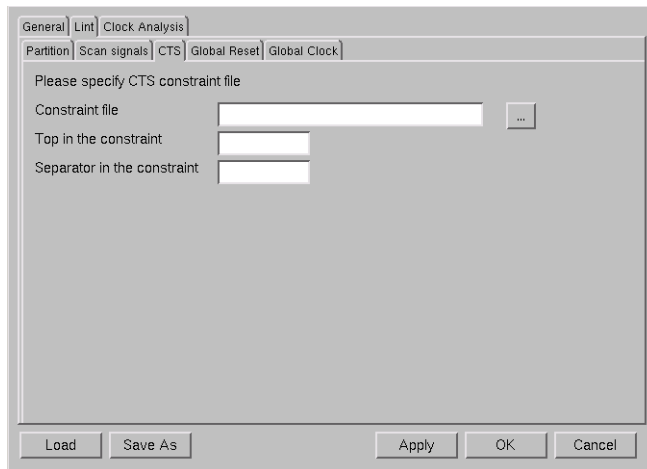


Figure: Project Setting -> Lint -> CTS Sub-tab

This option allows you to specify a CTS file to load from. It affects rule 25019 and 25021 checking.

The **Constraint file** text field allows you to specify a CTS file name.

The **Top in the constraint** text field allows you to specify the top entry when loading a CTS file.

The **Separator in the constraint** text field indicates the separator when loading a CTS file.

Global Reset Sub-tab

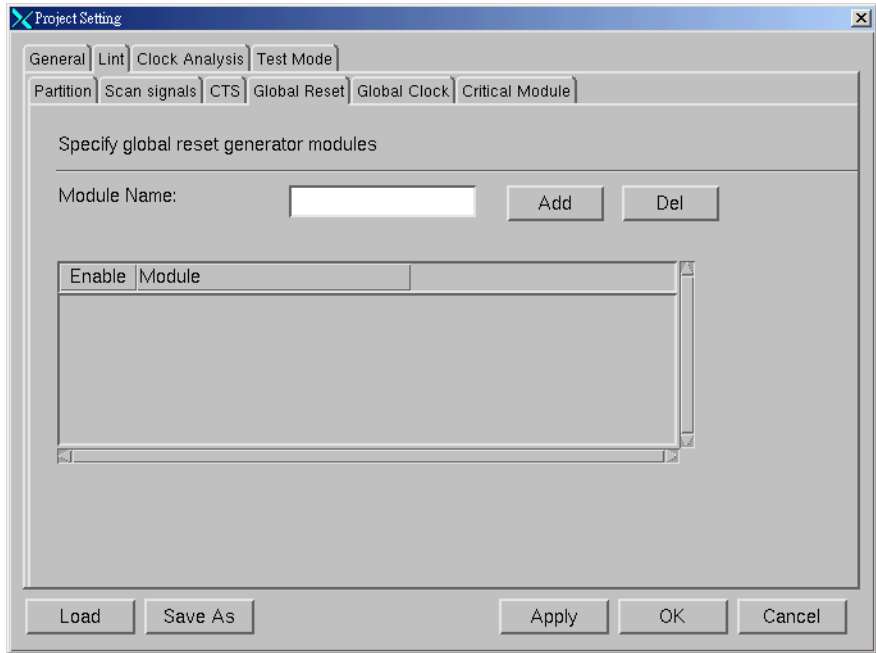


Figure: Project Setting -> Lint -> Global Reset Sub-tab

This option allows you to specify global reset generator modules. This option can also be set in batch mode: `"-reset_gen_module <module>"`.

Global Clock Sub-tab

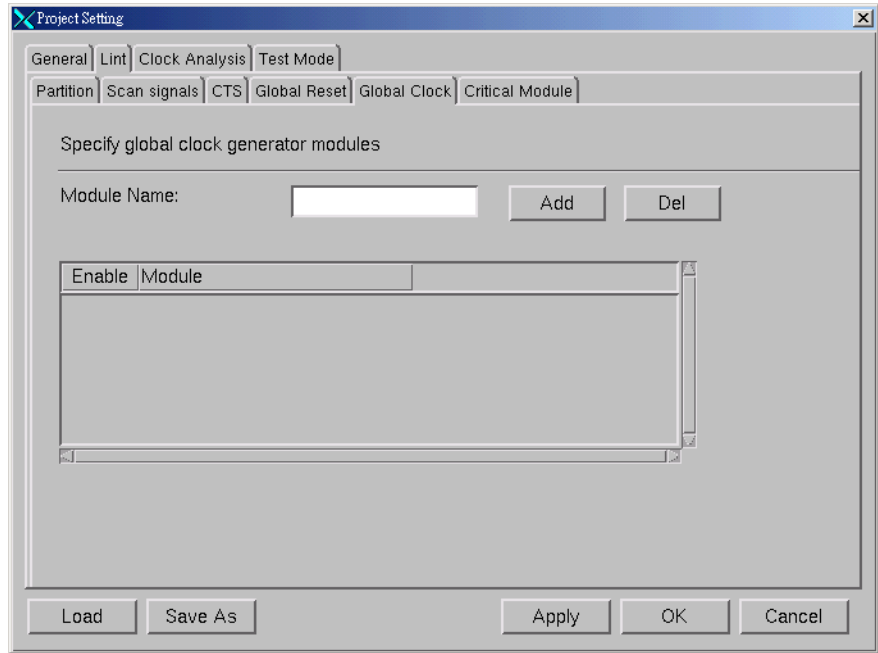


Figure: Project Setting -> Lint -> Global Clock Sub-tab

This option allows you to specify global clock generator modules. This option can also be set in batch mode: “-clock_gen_module <module>”.

Critical Module Sub-tab

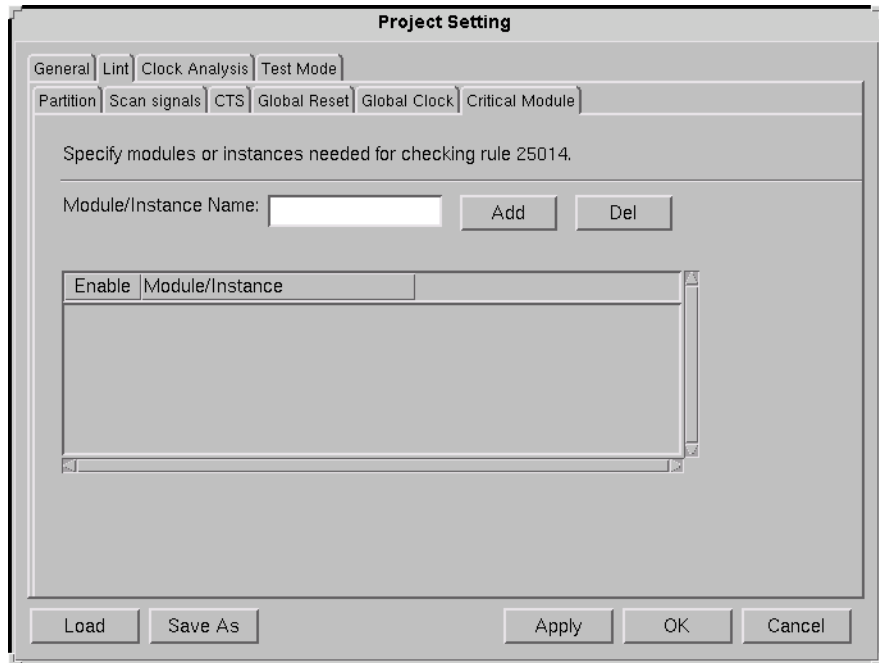


Figure: Project Setting -> Lint -> Critical Module Sub-tab

You can specify modules or instances to check rule 25014 in this sub-tab. Input a module or instance name in the **Module/Instance Name** field and click the **Add** button, the module or instance will be shown in the table below the **Module/Instance Name** field. There are the **Enable** and **Module/Instance** columns in the table. If you enable the checkbox in **Enable**, the selected module/instance will be checked. The default is enabled. The **Module/Instance** column shows the module or instance name. You can also use the **Del** button to remove the module or instance from the table.

Clock Analysis Tab

Unconcerned Path Sub-tab

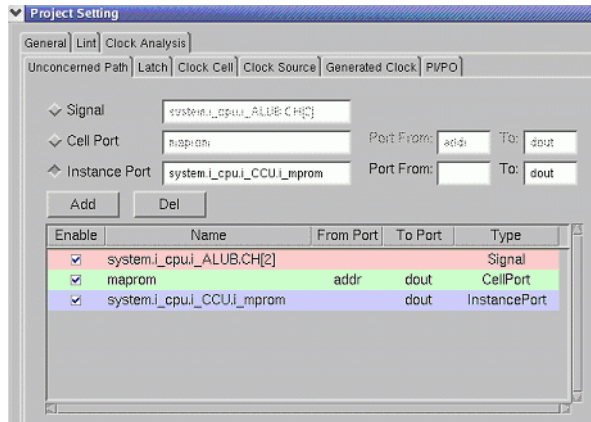


Figure: Project Setting -> Clock Analysis -> Unconcerned Signals Sub-tab

The **Unconcerned Path** sub-tab allows you to specify the unconcerned path when resolving clock source. When resolving a clock source, if some gates have more than one input, for example an AND gate, the resolving will be stopped because there is more than one choice to continue to trace backward. In this case, you can set one of the input signals as unconcerned signal. Thus, the resolving action will continue.

The **Signal** text box allows you to input the unconcerned signal. If a signal is unconcerned, when resolving the clock source, it will be ignored.

The **Cell Port** text box allows you to input a cell name together with the From Port and To Port of the cell. When resolving clock source, the path from output port of the cell to the input port of the cell will be ignored.

The cell **Instance Port** text box allows you to input a cell name together with the From Port and To Port of the cell instance. When resolving clock source, the path from output port of the cell instance to the input port of the cell instance will be ignored.

The **Unconcerned Path Name** text box is used to specify the signal/port name, which should be a full hierarchy name.

The **Add** button allows you to add a signal, cell or cell instance into the list.

The **Del** button allows you to delete a selected item from the list.

The checkbox in front of each row allows you to enable or disable the usability of the row.

Ignore Cell Sub-tab

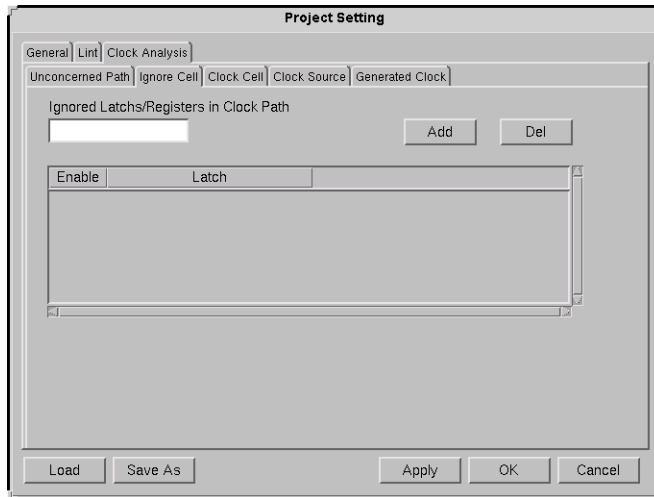


Figure: Project Setting -> Clock Analysis -> Ignore Cell

This option is used to specify the latch cells or registers which you like to pass through in clock extraction. The specified latch cells or registers in clock path will be ignored during clock extraction.

The **Add** button allows you to add a latch cell into the list.

The **Del** button allows you to delete a selected item from the list.

The checkbox in front of each row allows you to enable or disable the usability of the row.

Clock Cell Sub-tab

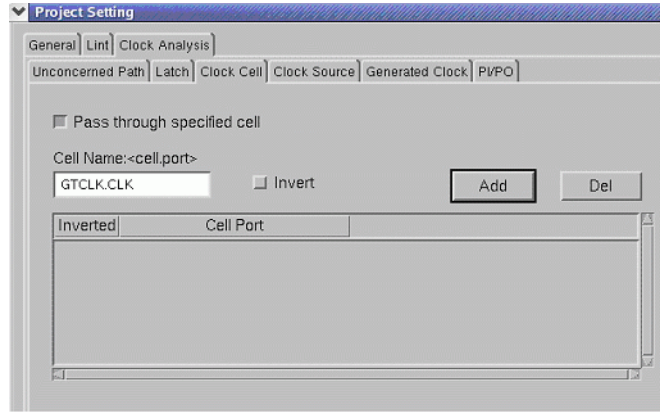


Figure: Project Setting -> Clock Analysis -> Clock Cell Sub-tab

The **Clock Cell** sub-tab allows you to specify the cell and its input port that can be passed through backward if the cell is encountered when tracing clock source.

The **Cell Name** text box allows you to input the cell and its clock port name together.

The **Invert** check box allows you to specify that the polarity of clock should be inverted or not when tracing through the specified port of the cell.

The **Add** button allows you to add a <cell.port> into the list.

The **Del** button allows you to delete a selected item from the list.

The checkbox in front of each row allows you to specify after passing through the cell.port, the polarity of the signal should be inverted or not.

Clock Source Sub-tab

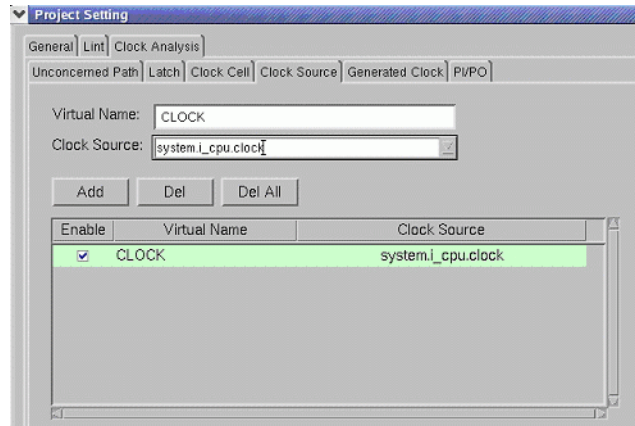


Figure: Project Setting -> Clock Analysis -> Clock Source Sub-tab

The **Clock Source** sub-tab allows you to specify clock source signal.

You can specify a clock source in the **Clock Source** comb box. And you can give it a virtual name in the **Virtual Name** text box. The virtual name of a clock source is optional. If a virtual name is specified, it implies the clock source comes from a virtual clock source. There could be more than one clock source comes from same virtual clock source.

If clock source tree is extracted, all the current roots of the clock source tree will be listed in the **Clock Source** comb box for you to choose. Or you can input any full hierarchy signal name or full hierarchy cell instance and its port as the clock source.

To add a specified clock source with port/signal name and source type in the list window, use the **Add** button. The **Remove** button deletes a selected row from the list window while the **Remove All** button deletes all.

The checkbox in front of each row allows you to enable or disable the usability of the row.

Generated Clock Sub-tab

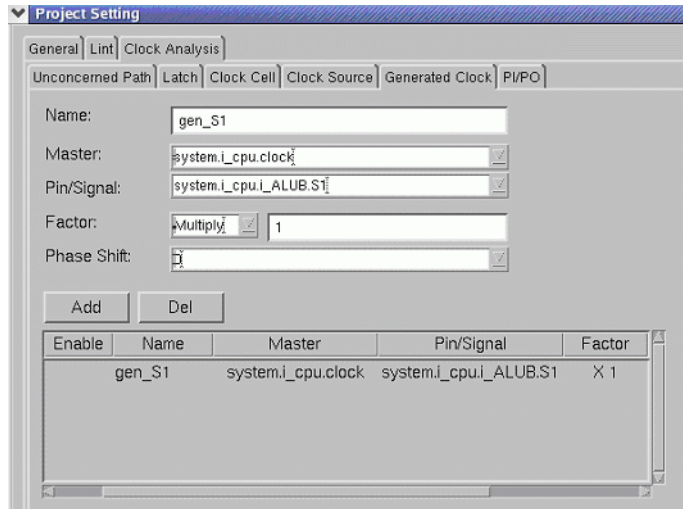


Figure: Project Setting -> Clock Analysis -> Generated Clock Sub-tab

The **Generated Clock Source** sub-tab allows you to specify generated clock source.

The **Name** text box allows you to give a name to the generated clock source. It is optional.

The **Master** text box allows you to specify the master clock source where the generated clock source derived from.

The **Pin/Signal** text box allows you to specify the generated clock source. It should be a full hierarchy signal or a full hierarchy cell instance port.

The **Factor** comb box allows you to specify the frequency relationship between generated clock source and the master clock source. You can choose either **Divide** or **Multiply**, followed an integer.

Divide <integer> means that the frequency of the generated clock source can be obtained by dividing the frequency of the master clock source by specified <integer>. Multiply <integer> means that the frequency of the generated clock source can be obtained by multiplying the frequency of the master clock source by specified <integer>.

The **Phase Shift** box allows you to specify the phase shift between the generated clock source and the master clock source. You can choose phase shift 0, which means no phase shift; 90, which means T/4 phase shift; 180, which means T/2 phase shift; 270, which means ?T phase shift. Here T means the clock cycle.

The **Add** button adds an item into the grid box.

The **Del** button removes the selected items from the grid box. When removing the virtual clock source item, all containing clock source items should be removed first.

Test Mode Tab

The **Test Mode** tab includes the **Constants** and **Nets** sub-tabs.

Constants Sub-tab

Please refer to the **Project Setting -> General -> Constants** Sub-tab for more details.

Nets Sub-tab

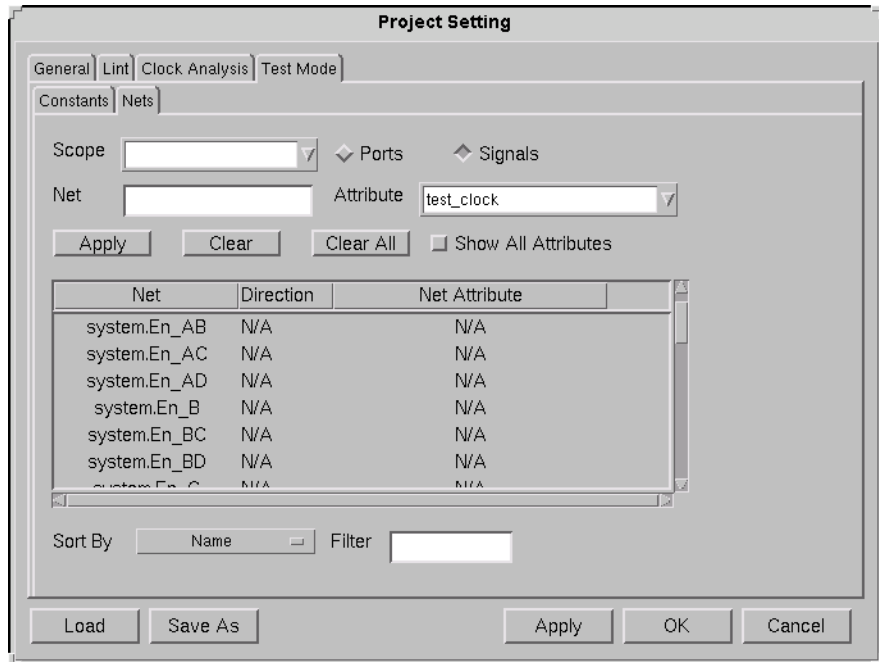


Figure: Project Setting -> Test Mode -> Nets Sub-tab

You can use options in this tab to define ports or signals as test pins in any scopes of the design.

In the **Scope** field, you can input a scope to define test pin. The scope only can be a module or an instance. If the scope is empty, nets of all top modules will be listed in the net display area. This field is blank by default.

You can use the **Ports** and **Signals** buttons to set nets of your interest. Enable the **Ports** button to view all ports in the specified scope. Enable the **Signals** button to view all signals in the specified scope.

The **Net** field shows the port or signal name selected in the net display area.

The **Attribute** selection field lists all test mode types, including “test_mode_inverted”, “test_clock”, “test_scan_enable”, “test_scan_enable_inverted”, “test_asynch” “test_asynch_inverted” and “test_mode”. You can select one attribute option and set it for the specified net in the **Net** field.

If you have set test mode for nets, enable the **Show All Attributes** option to view the specified nets in the net display area. When the **Show All Attributes** option is enabled, the **Scope** and **Net** fields will be invalid and don’t allow any changes. When the **Show All Attributes** option is disabled, nets belonging to the current scope will be shown in the net display area. The **Scope** and **Net** fields will be valid and allow any changes.

Click the **Apply** button to set the test mode attribute of the selected net in the net display area. Click the **Clear** option, the attribute settings of the selected net(s) will return to default value, which is N/A. Click the **Clear All** option, the attribute settings of all net(s) will return to default value, which is N/A.

You can use the **Sort By** option to view the nets in the net display area, based on the criteria of net name, direction and attribute.

You can input a string in the **Filter** field. Items matching the string will be shown in the display area and the rest are filtered. Note that if scope or net type is changed between ports and signals, the filter function will not work. After you click the **Apply** button in the lower right corner, the matched nets will be showed in the net display area.

Create Domain

Toolbar Button:



This command creates a clock source tree. You can create a clock source tree repeatedly before or after linting a design.

Compile

This command compiles your design even if the design has been compiled into memory. Each time you select the command, the compilation process will be activated.

Lint

Toolbar Button:



This command lets you start the checking process.

If the design has not yet been compiled into memory, nLint will compile and lint the design. If the design has already been compiled into memory, nLint will check if there is any modification after design compilation. The design will be re-compiled if it has been modified.

A Report Viewer will appear after the checking process. The rule checking process will be run based on the rule setting file (*rs* file) you defined under the Lint or Administration mode. Notice that you must switch back to Lint mode when you lint the design.

Before linting, if there is no clock domain information, the related rules of clock domain will not be checked.

Local Lint

Toolbar Button:



You can invoke this command for local-lint. Please note that you should enable the **Compile design to library (Note: Before 'Local-lint', the option should be checked.)** option in the **Preferences -> Lint tab -> General** tab, nLint will compile the design and save the result to the specified library, so the directory will be valid to vericom and writable. If no library is specified, the work.lib++ in work directory will be the default library. You can also use the command line, "-lib" to specify it.

Before you invoke this command, you can specify a vericom library directory in the **Vericom Library** option in the **Project Setting** command -> **General** tab -> **Lint Entry** tab, so that nLint will compile design incrementally and save the result to the specified library.

Tools Commands

Find Scope

This command can easily locate source code that user is looking for. It is available only when the design imported is compiled or the hierarchy tree of the design is generated.

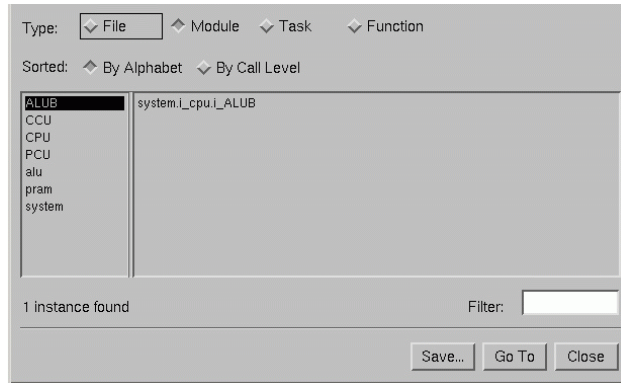


Figure: Tools -> Find Scope

You can choose **File**, **Module**, **Task** or **Function** to find.

A file, module, task or function can be sorted **By Alphabet** or **By Call Level**.

The left side window displays file, module, task or function that you look for. For example, if you want to find a specific module, all possible modules of design show up in the left window. In the right window, instances of the declaration are listed.

The **Filter** is used to filter out unmatched pattern in the declaration list window. For example, in this case, if you key in "A*" with **Return Key**, only the ALUB is left in the declaration list window. Here wildcard "*" is supported. After you empty all contents in the filter field, click **Return** key, the filter function is disabled.

After an item listed in the declaration list window is chosen, you can click **Go To** for the declaration in source code edit window.

The **Save** button allows you to specify a file name to save the highlighted declaration item with its instances.

Test Regular Expression

This command allows you to test the regular expression is written as expected or not.

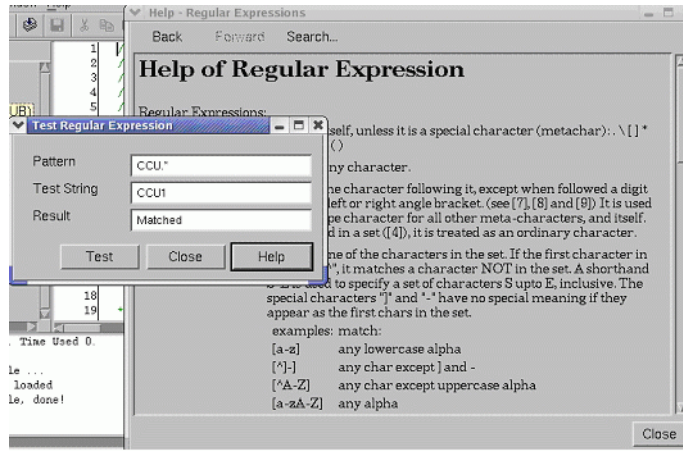


Figure: Tools -> Test Regular Expression

You can put the regular expression you want to test in the **Pattern** text box. And put any string to be tested in the **Test String** text Box. Click **Test** button, you will get the result in the **Result** text Box. The result will show you the testing is **Matched** or **Not Matched**. Click **Close** to close this form. Click **Help** to pop up the help information for the regular expression.

Debussy

This command lets you run the Verdi system. This command is disabled if the project is empty.

Please specify the program and command options of the Verdi system with the **Preferences -> Debussy/Verdi** command. When the command is invoked, in fact nLint tries to fork a process to run the command and command options specified.

NOTE: Make sure that you have set the correct path to invoke the Verdi system.

Rule Organizer

This command brings up the Rule Organizer. For details, refer to [Rule Organizer](#).

Report Viewer

This command brings up the Report Viewer. For details, refer to [Report Viewer](#).

Preferences

This command defines the settings of font or color for each item in nLint, and sets the correspondence between nLint and Verdi. You can customize the way to extract clock domain in nLint, to lint and to show violations, etc.

General Tab

In this tab, the general options for nLint are set. The options will affect all function in nLint.

Option Sub-tab

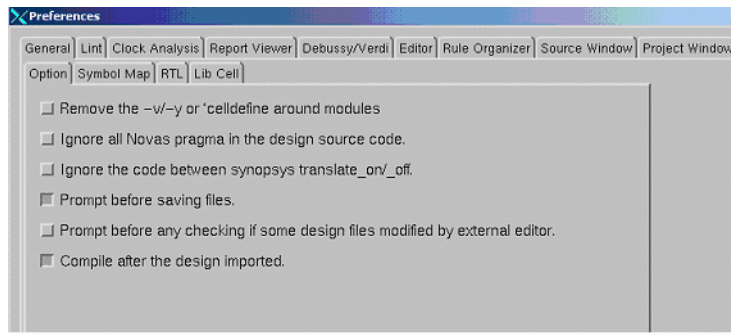


Figure: Preferences -> General -> Option

The **Option** sub-tab is used to specify some general option for the tool.

Remove the -v/-y or `celldefine around modules: this option equals command line option -ssv/-ssy/-ssz. It indicates all the library cell defined by -v/-y or `celldefine becomes normal hierarchy module.

Ignore all Novas pragma in the design source code: this option equals command line option -ignore_pragma. It is used to ignore the Novas pragma embedded in design source code.

Ignore the code between synopsys translate_on/off: this option tells compiler to treat the source code embraced by synopsys directive translate_on/off to be comment.

Prompt before saving files: this option to indicate the tool to prompt message before saving modified source file.

Prompt before any checking if some design files modified by external editor: if some source files are opened by nLint editor and it is modified by other editor at the same time, nLint will detect the modification. When lint command is click, nLint will prompt message to ask you whether to re-compile the modified file or just lint based on current compiled design data base.

Compile after the design imported: this option indicates the tool to compile the design once the files are loaded into the tool.

Symbol Map Sub-tab

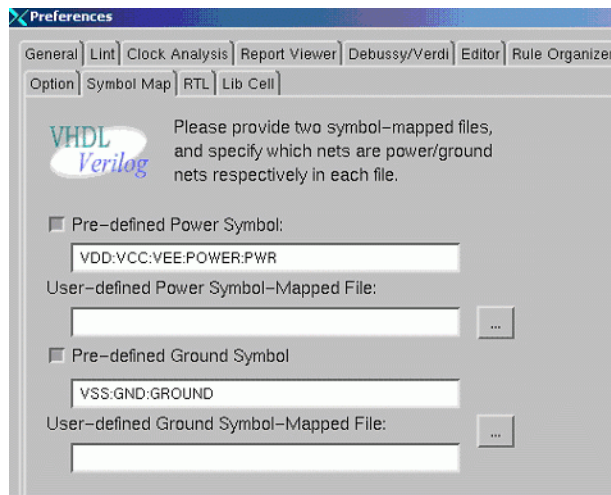


Figure: Preferences -> General -> Symbol Map

The **Symbol Map** sub-tab used to indicate that which signals in design are ground signal and which signals in design are power signal.

The power or ground signals can be specified by the **Pre-defined...** text box or a file. The file format is as follows.

```
VCC Power
VDD Power
VDD1 Power
GND Ground
GND1 Ground
```


RTL Sub-tab

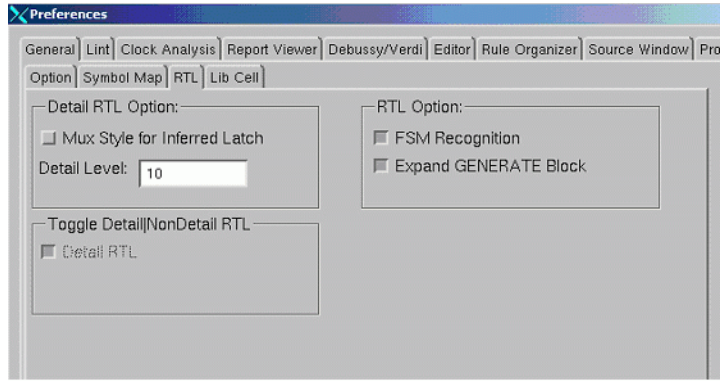


Figure: Preferences -> General -> RTL Sub-tab

The **RTL** sub-tab allows you to customize the RTL extraction process. It is the same as the Verdi system. But in nLint, we force to turn on Detail RTL extraction.

Lib Cell Sub-tab

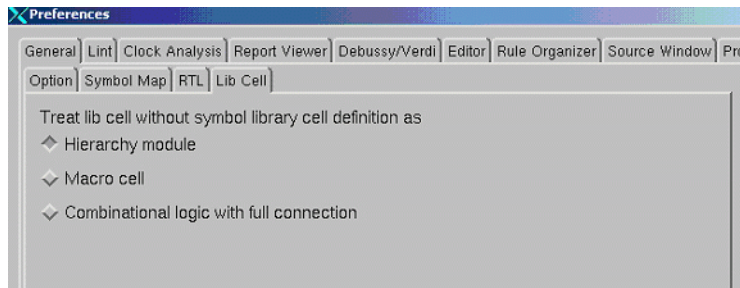


Figure: Preferences -> General -> Lib Cell

The **Lib Cell** sub-tab indicates how nLint treats the library cells. Library Cell in nLint are those modules specified by `-v/-y` option or those cells embraced by ``celldefine`/`endcelldefine` in source code. There might be function description in those modules and the function description is possible to be encrypted. If symbol definition is provided for those cells by setting two environment `TURBO_LIB_PATHS` and `TURBO_LIBS`, nLint will follow the symbol definition. But if the symbol definition is not provided, nLint will handle them with alternatives.

- *Hierarchy module*: nLint will treat them as normal module and look into the function description of the library cell.

- *Macro cell*: nLint will treat them as black box (or Macro cell), just like the -bb command line option. In this way, nLint will not look into the function description of the cell even there is.
- *Combinational logic and full connection*: nLint will not look into the function description of the cell and treat them as a combinational cell with input-output fully connected.

Lint Tab

This tab is used to specify option about linting.

The **Lint** tab contains the following sub-tabs, **General**, **RS**, **Library**, **Violation number**, **Severity** and **Message**, which allow you to customize the preference settings associated with the checking process. These items are related to the command line options under batch mode. You may set the number of warnings, severity string and message format, etc.

General Sub-tab

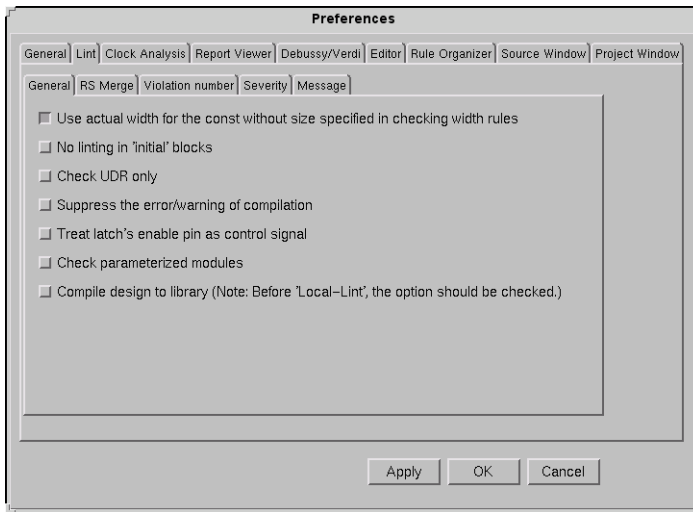


Figure: Preferences -> Lint -> General Sub-tab

The **General** sub-tab is used to specify general option for linting.

Use actual width for the const without size specified in checking width rules: This option equals to -actualwidth.

No linting in 'initial' blocks: This option equals -ignore_initial. It tells the tool not to check rules in initial block for Verilog design.

Check UDR only: This option equals `-udr_only`. It tells the tool to check user defined rules only.

Suppress the error/warning of compilation: This option equals to `-no_syntax`. It tells the tool to skip all compilation errors/warning.

Treat latch's enable pin as control signal: nLint treats "latch enable" as clock by default. When this option is on, nLint treats "latch enable" as control signal. Enable this option to keep latch from checking related rules: 22053, 22054, 22055, 22127, 22128, 22129, 22130, 22131, 22134, 22225 and 22231. This option is off by default.

Check parameterized modules: This option allows you to check all parameterized modules. If one module is instantiated with different parameters, nLint will treat them as different modules and check all of them.

Compile design to library (Note: Before 'Local-lint', the option should be checked.): When the option is *on*, nLint will invoke "vericom -smartinc" to export library to the disk for doing lint or local-lint. The design result will be compiled and saved to the specified library, so the directory will be valid to vericom and writable. When the option is enabled, a library will be exported to disk before doing lint or local-lint. The option is OFF by default.

In GUI mode, if the design is imported from library, this option will be disabled. In batch mode, you can use "`-vericom [on|off]`" to specify this option.

RS Merge Sub-tab

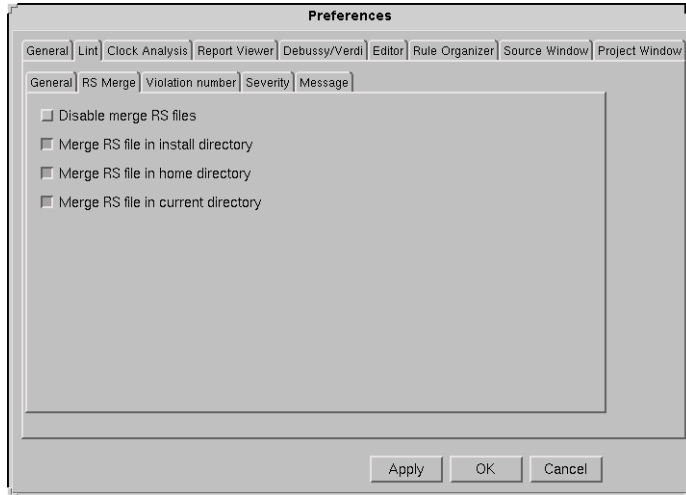


Figure: Preferences -> Lint -> RS Merge Sub-tab

The options in the **RS Merge** sub-tab are used to indicate the way to merge RS files, just like the command line options `-nors` and `-drs`.

Violation Number Sub-tab

You can enter a number in the text field of **Stop linting when [] warnings have been encountered** to specify the number of warnings.

You can enter a number in the text field of **Maximum number of warnings per rule** to specify the maximum number.

The selection filed of **Check rules between minimum severity and maximum severity** allows you to specify level of severity.

The text field of **Stop linting when compilation error exceeds []** allows you to specify a number. The default number is 200. The default is ON.

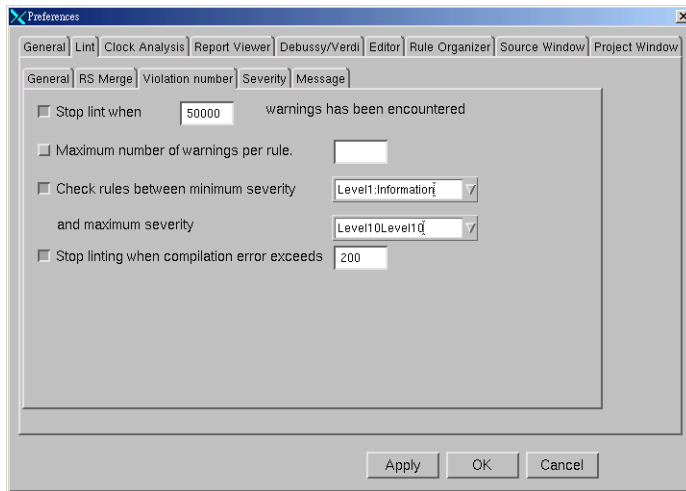


Figure: Preferences -> Lint -> Violation number Sub-tab

Severity Sub-tab

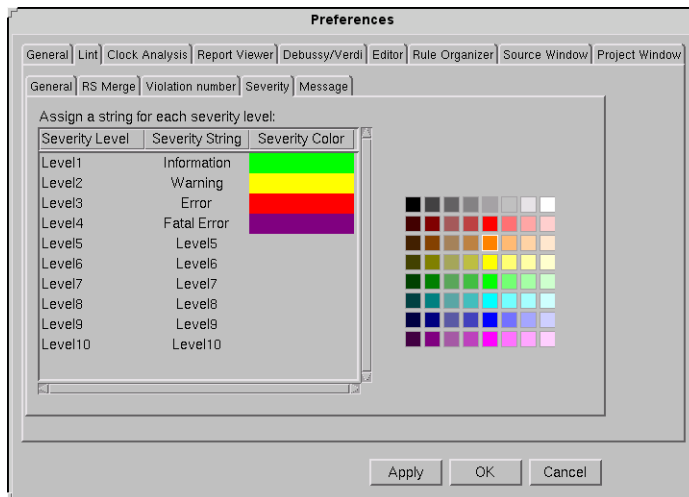


Figure: Preferences -> Lint -> Severity Sub-tab

The **Severity** sub-tab allows you to specify a severity string and a severity color for each severity level. By default there are 10 severity levels in nLint. The severity color shows different degrees of severity and the severity color is displayed in violation line number of the source code pane of nLint Project Window. By default, the color for level 1 is green; level 2 is yellow; level 3 is red and level 4 is purple.

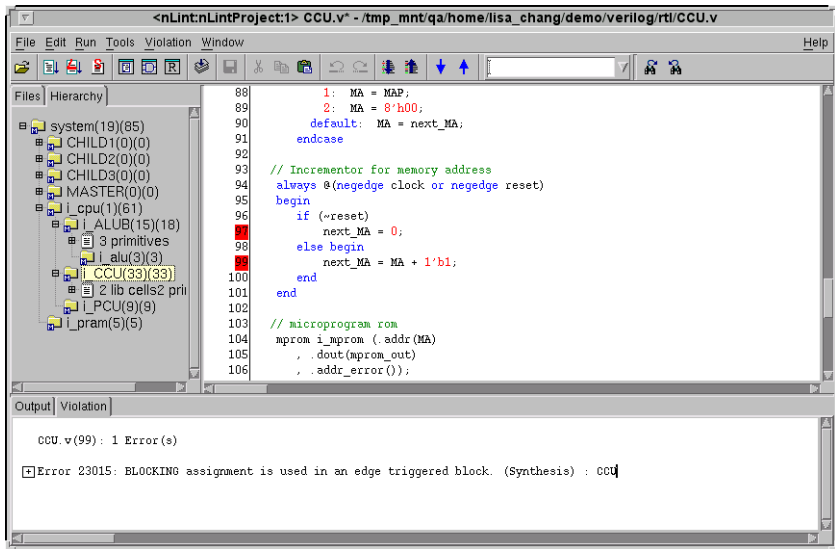


Figure: Example of Severity Color Shown in Violating

Message Sub-tab

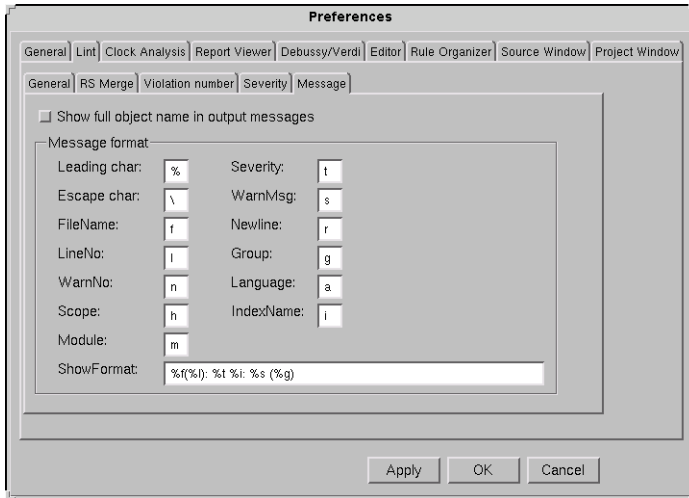


Figure: Preferences -> Lint -> Message Sub-tab

Show full object name in output message allows you to specify not to abbreviate the string for the violated object.

In the **Message Format** section, you can also configure the format of error messages when you export the errors to an ASCII file. Except **ShowFormat**, all

other fields are read-only. The nLint **Message Format** defines the meaning of the character and can be used in the **ShowFormat**. All character should be led by the Leading char '%'. You can use **Escape char** to print out the original character if they are used to define the format.

After the **ShowFormat** is specified, nLint will follow the configuration to output warning message to the ASCII file.

Clock Analysis Tab

The **Clock Analysis** tab contains the **General** and **Pass Through** settings that are associated with clock domain extraction.

General

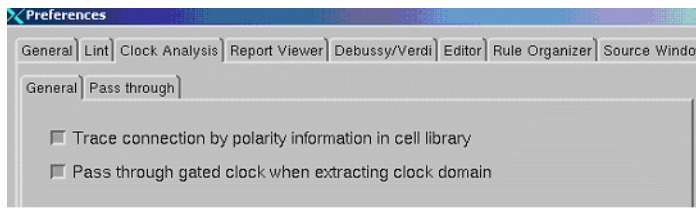


Figure: Preferences -> Clock Analysis -> General Tab

The **General** tab allows you to customize the resolving in clock source tree extraction.

Trace connection by polarity information in cell library allows you to indicate the tool to trace connectivity by the polarity information provided by symbol library, which is translated from cell library.

Pass through gated clock when extracting clock domain allows you to guide the tool to merge all the gated clock domains into one clock domain.

Pass Through

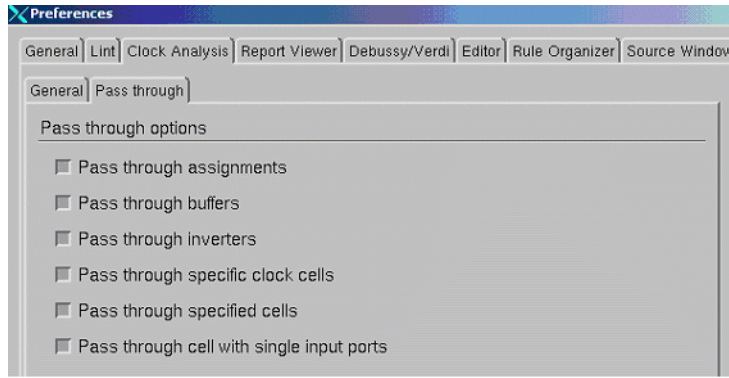


Figure: Preferences -> Clock Analysis -> Pass Through Tab

The **Pass through** tab allows you to specify pass through what kind of logic when extracting clock domain. The setting will impact the clock source signal of a clock domain. But it will not impact the clock source tree.

- The **Pass through assignments** checkbox allows you to indicate nLint whether to pass the assignment like "assign a = b" when tracing backward of the clock signal for extracting clock domain. By default, it is ON.
- The **Pass through buffer** checkbox allows you to indicate nLint whether to pass the buffer when tracing backward of the clock signal for extracting clock domain. By default, it is ON.
- The **Pass through inverters** checkbox allows you to indicate nLint whether to pass the inverters when tracing backward of the clock signal for extracting clock domain. By default, it is ON.
- The **Pass through specified clock cell** checkbox allows you to indicate nLint whether to pass the clock cell in symbol library when tracing backward of the clock signal for extracting clock domain. By default, it is ON. A clock cell is the symbol cell with cell type as clock cell, and input clock pin, output clock pin.
- The **Pass through specified cell** checkbox allows you to indicate nLint whether to pass the cell, which is specified in **Project Setting -> Clock Analysis -> Clock Cell** by the input port specified together, when tracing backward of the clock signal for extracting clock domain. By default, it is ON.
- The **Pass through cell with single input port** checkbox allows you to indicate nLint whether to pass any cell with unknown type but with only

single input port when tracing backward of the clock signal for extracting clock domain. By default, it is ON.

Report Viewer Tab

The **Report Viewer** tab contains the following sub-tabs, **General**, **Detail message**, **Clock Domain**, **Format** and **Color**, which allow you to configure your report output.

General

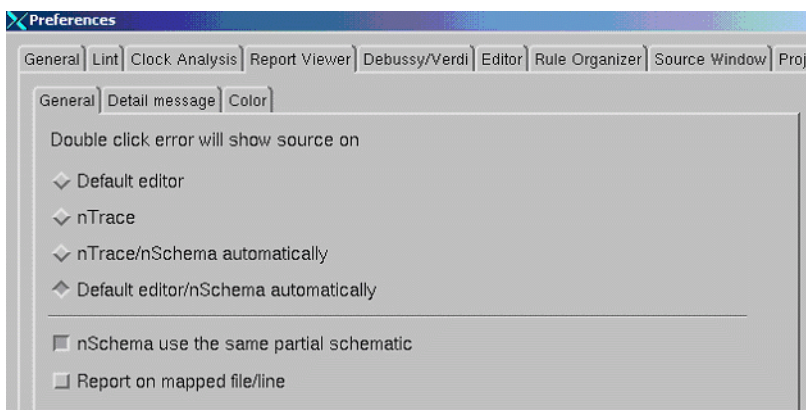


Figure: Preferences -> Report Viewer Tab -> General

Under the **General** sub-tab, you may set the tracing tool for source code when double-clicking the errors detected by nLint in report viewer. There are four choices:

- **Default Editor:** use of preferred editor, which is set in the Editor tab.
- **nTrace:** show the source code in *nTrace* window.
- **nTrace/nSchema automatically:** show the source code in *nTrace* or *nSchema* window, which will be automatically decided by nLint. The rule error with path information will be shown in *nSchema* window, i.e., gated clock in DFT group.
- **Default Editor/nSchema automatically:** show the source code in the default editor or *nSchema* window.
- **nSchema use the same partial schematic:** show to the same *nSchema* window if it is checked.
- **Report on mapped file/line:** in verilog 2001, the line compiler directive is introduced. We support this except the feature level. When it turns on, nLint

will try to show the violations that the original source file/line mapped, based on the line compiler directive. Also when exporting violations to ASCII file, nLint will export the original file/line of the violations according to this setting.

Detail Message

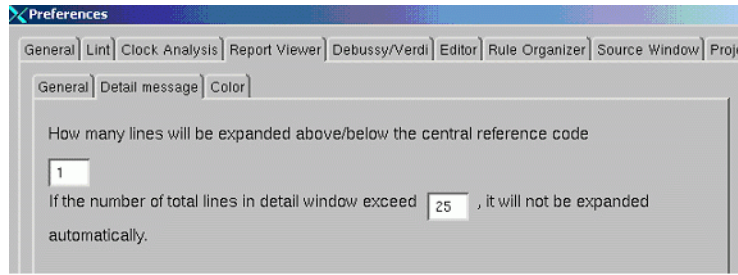


Figure: Preferences -> Report Viewer Tab -> Detail message

The **Detail message** sub-tab allows you to set the line number to expand when during showing the source code in detail window of report viewer.

The first number is used to specify number of lines be expanded. As demonstrated in the following figure, above and below the central reference code line, which is the blue highlighted line, nLint expands one more line to show the piece of source code.

```
ALU6 w(67): output signal "ALU" should not be referenced inside the module
 assign net_2... (ALU6 w)
56
57 assign net_2 = (ALU == 0'100) & (~CH[1]);
58 assign net_3 = S1 & (~IR[0]);
 MUX (MUX w)
44 output S_;
45 output [7:0] ALU;
46 output [7:0] IR;
```

If there are too many lines shown in detail window of report viewer, it is not good to expand the reference source code by default. The second number is used to set the threshold. As demonstrated in the following figure, contents in the detail window are up to 25 lines. So the source code reference is not expanded by default.

```

TopModule v(55): combinational loop detected on signal 'ALU'
[+] system_i_cpu.ALU (TopModule.v)
  system_i_cpu.ALU -> system_i_cpu.ALU (TopModule.v)
  ALU[7:0] -> i_pcc.ALU[7:0] (TopModule.v(59))
[+] PCU:Always6#Always5:61:89:Mix -> IDB[7:0] system_i_cpu.i_pcu (PCU.v)
  IDB[7:0] -> system_i_cpu.IDE[7:0] (TopModule.v(64))
  TMR[7:0] -> i_pcc.TMR[7:0] (TopModule.v(74))
[+] ALU:Always8#Always7:75:82:Mix -> X1[7:0] system_i_cpu.i_alu (ALU.v)
  X0[7:0] -> i_alu.X[7:0] (ALU.v(78))
[+] X[7:0] -> alu:Always4#Always3:45:61:Sub system_i_cpu.i_alu.alu (alu.v)
[+] alu:Always14#Always10:49:61:Mix -> out[7:0] (alu.v)
  out[7:0] -> system_i_cpu.i_ALU.ALU[7:0] (ALU.v(73))
  ALU[7:0] -> system_i_cpu.ALU[7:0] (TopModule.v(64))

occ_aloo:

```

Color Sub-tab

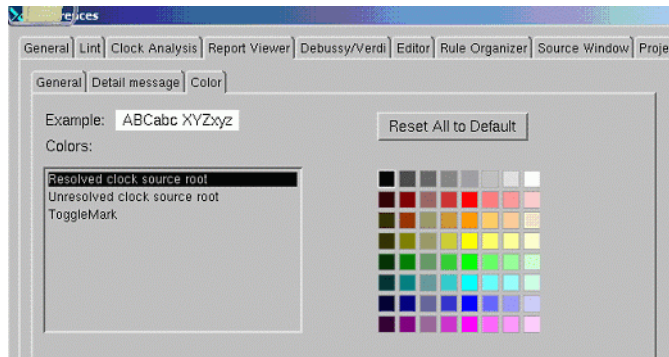


Figure: Preferences -> Report Viewer -> Color Sub-tab

From the **Color** sub-tab, you can also choose the color for "Unconcerned clock domain", "Unresolved clock domain" and "Violations marked" in report viewer.

Debussy/Verdi Tab

The **Debussy/Verdi** tab allows you to specify Debussy/Verdi execution command with argument.

Editor Tab

The **Editor** tab allows you to set your favorite editor with parameters when tracing the source code.

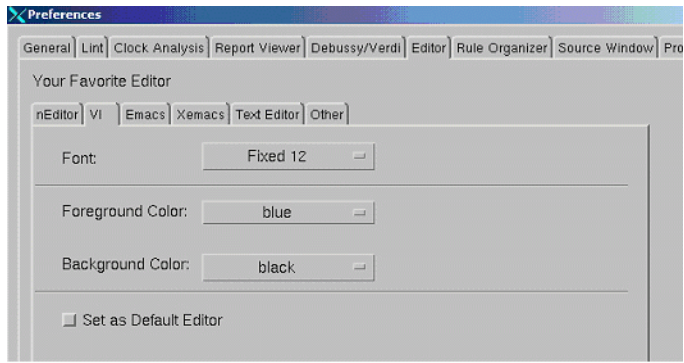


Figure: Preferences -> Editor Tab

Rule Organizer Tab

The **Rule Organizer** tab allows you to configure the displayed columns in the rule list window, the right part of the Rule Organizer. You may show or hide the columns, and organize the order of columns displayed in the rule list window.

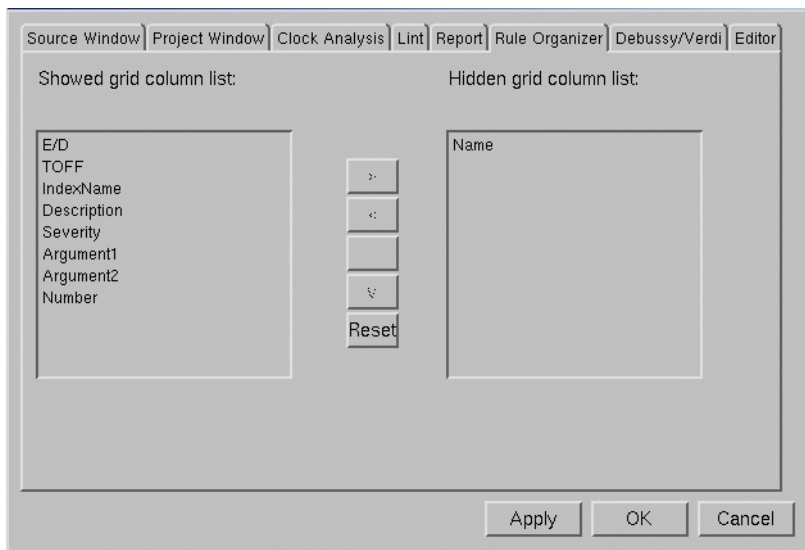


Figure: Preferences -> Rule Organizer

Source Window Tab

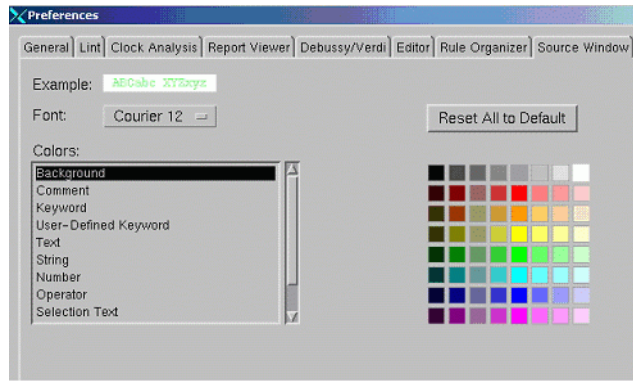


Figure: Preferences -> Source Window Tab

The **Source Window** tab allows you to set your favorite font and color for the background, comment, keyword, text, string, etc.

Project Window Tab

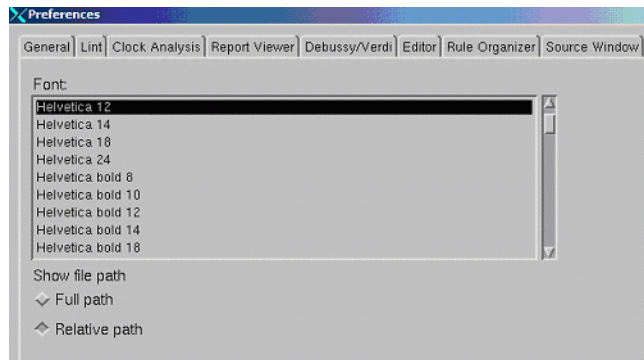


Figure: Preferences -> Project Window Tab

The **Project Window** tab allows you to set your preferred font and display file path for the Project Window.

Violation Command

Next Violation Line

Bind Key: F5

This command highlights the next violation line in the source code pane of nLint Project Window.

Previous Violation Line

Bind Key: Ctrl+F5

This command highlights the previous violation line in the source code pane of nLint Project Window.

Next Violation

Bind Key: F6

This command highlights the next violation in the **Violation** tab of the bottom pane in nLint Project Window. This command is also available in the click-right option menu when you click on a violation in the **Violation** tab in the bottom pane of nLint Project Window.

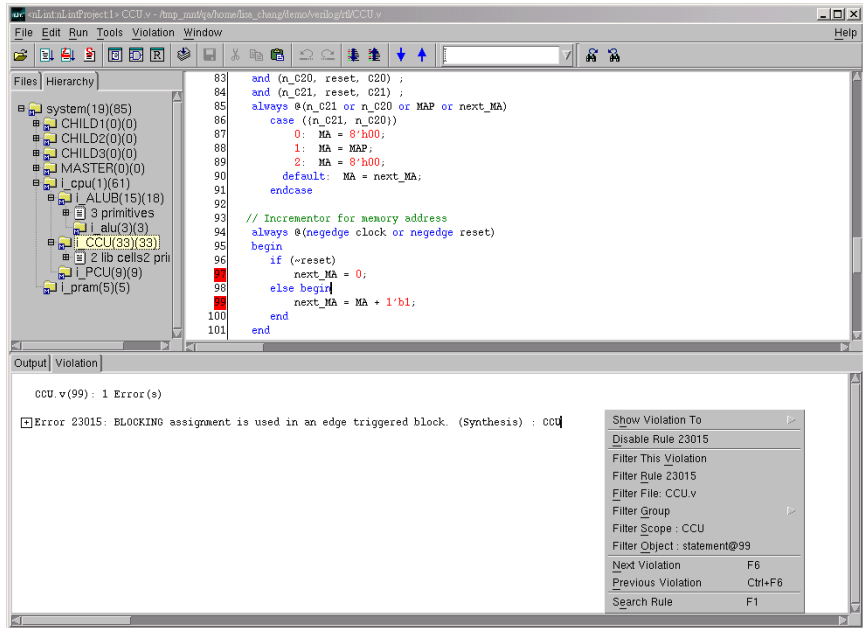


Figure: Example of Click-right Option Menu of Violation

Previous Violation

Bind Key: Ctrl+F6

This command highlights the previous violation in the **Violation** tab of the bottom pane in nLint Project Window. This command is also available in the click-right option menu when you click on a violation in the **Violation** tab in the bottom pane of nLint Project Window.

Window Command

Output Window

This command brings up the Output Window with the last checking results.

Help Commands

Contents

This option brings up a Help Window. You can get help from the **Contents**, **Index** or **Search** tab.

Rules Reference

This option brings up a Help Window for the Rule Category. You can get help from the **Contents**, **Index** or **Search** tab.

About

This choice offers information about the current nLint release, such as the current release's version and date.

Rule Organizer

Overview

The Rule Organizer is invoked when you select the **Tools -> Rule Organizer** command in nLint main frame window. This Rule Organizer is used mainly for three purposes: 1) to enable or disable checking certain rules, 2) to configure the arguments used in certain rules, and 3) to provide editing functions for organizing rules.

- **Compilation & Elaboration**, this is group of syntax rules. They are checked by compiler. And they cannot be re-organized. This group is always enabled.
- **nLint**, this is the default group of rules provided.
- **RMM**, this is the RMM compatible rules organization.

The checking rules in the default top group 'nLint' of this Rule Organizer are categorized in the following ten groups:

Simulation: [Enable] Rule violation in this group will degrade the speed of simulator or conduct an unexpected simulation result.

Synthesis: [Enable] Rules in this group check the language elements, i.e., which synthesis tool is not supported; which synthesis tool cannot construct hardware absolutely under the semantics of description by user; which will directly cause the mismatch of simulation result between pre-synthesis and post-synthesis.

DFT: [Enable] This group includes the rules that must be satisfied for testing, especially for the successful use of ATPG tools.

ERC: [Enable] This group includes the rules checked in the net-list before or post P&R. All the rules are focus on connection, like input floating and output floating.

Design Style: [Enable] Rules in this group will check the potential errors in connectivity, clock/reset signals.

Language Construct: [Enable] This group offers a powerful semantics checking. There might be potential errors in design if there is any violated rule in this group.

HDL Translation: [Enable] Rules in this group check the translation problem between Verilog and VHDL.

Coding Style: [Disable] Rules in this group define the style in coding. Good coding style increases readability.

Naming Convention: [Disable] Rules in this group define the naming convention in coding. Readability will be increased if they are considered.

VITAL Compliant: [Disable] Rules in this group check for VHDL code to see if it is VITAL compliant.

Clock: [Enable] Rules in this group check for clock usage scheme in design.

Block Interconnect: [Enable] Rules in this group check the problems when assembling blocks into the design.

NOTE: Every time you run nLint, you can specify the active groups of rules for nLint checking. You can also configure the arguments by using the **Rules -> Properties** command.

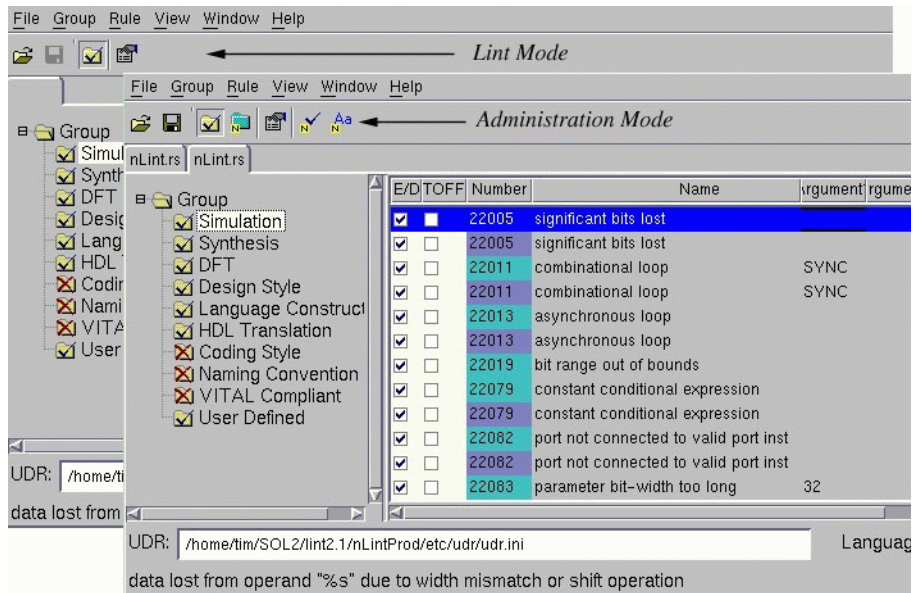


Figure: Rule Organizer Under Lint Mode (Rear) and Administration Mode (Front)

Rule Organizer provides two working modes: Lint mode and Administration mode.

- **Lint Mode:** Show the merged RS file only. Initially, there is no filename applied to the merged RS file except that the Disable merge RS files option in the *Open RS Files* form is turned on when you load the selected RS file(s) listed in the selection window. You may save the information on this tab or rename the merged RS file with the Save As command and lint your design

later with this RS file. Notice that in the rule property dialog box, you can only modify the configurable options and arguments under this mode.

- **Administration Mode:** Show all loaded RS files (with filename) each on a tab. There is no merged RS file shown under this mode. You may modify the configurable options and arguments; you are also allowed to configure the settings of rule options and arguments by clicking the **Argument non-configurable** and **Status non-configurable** options; and you also can configure the setting of UDR file. Notice that you are required to switch back to the Lint Mode before linting the design.

The menu commands in Rule Organizer are described in the following sections.

File Commands

Open

Toolbar Button:



This command opens the *Open RS/CSV Files* form where you can set multiple rule-setting files (*.rs *.csv) once at a time. nLint will merge all RS/CSV files that it finds in the pre-defined directories and uses the merged result for checking.

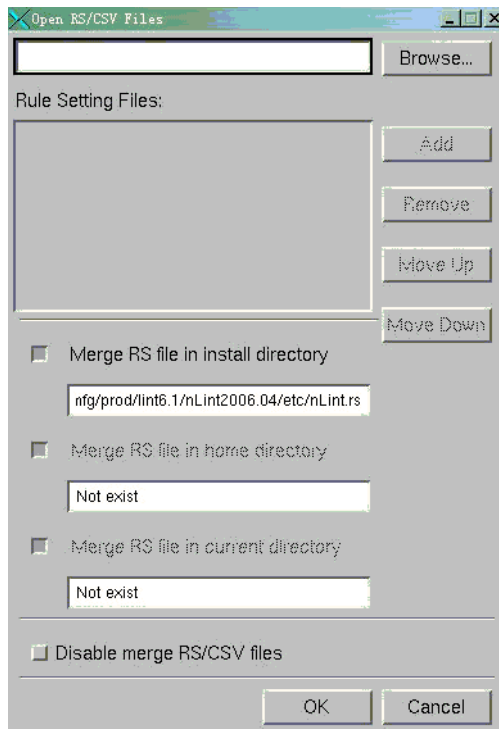


Figure: Open RS/CSV Files Form

You may select the desired RS/CSV files by clicking the **Browse** button. To append the selected RS/CSV file into the **Rule Setting Files** selection window, click the **Add** button. To delete the unnecessary RS/CSV file from the selection window, click the **Remove** button. You may also move the order of selected rule files upward and downward by using the **Move Up** and **Move Down** buttons.

There are four options for the integration of RS files:

- When the **Merge RS file in install directory** option is *on*, the RS file in the install directory will be merged while generating the merge RS file.
- When the **Merge RS file in home directory** option is *on*, the RS file in the home directory will be merged while generating the merge RS file.
- When the **Merge RS file in current directory** option is *on*, the RS file in the current directory will be merged while generating the merge RS file.
- The **Disable merge RS/CSV files** option controls the merging of the above RS/CSV files. When this option is *off*, the RS/CSV files in the above directories and the selected RS/CSV files in the selection window will be merged and loaded to Rule Organizer when you click **OK**. These RS/CSV

files will be merged one by one from top to bottom; the order is <RS file under install directory>, <RS file under home directory>, <RS file under current directory> and the RS/CSV files in the selection window. If you turn on the option, nLint will only load the RS/CSV file(s) selected in the selection window.

When open RS/CSV files, Rule Organizer will automatically search for the <nLint_install_dir>/etc/udr directory. If there are *udr.ini* and *action.tcl* under the directory, the Rule Organizer will automatically load the *udr.ini* file as the UDR file. Switching the Lint and Administration modes allows you to refresh the contents in the *udr.ini* or *action.tcl* file. When switching from Administration mode to Lint mode, the Rule Organizer will reload all RS/CSV files with the UDR file.

Save

Toolbar Button:



Bind Key:

CTRL+S

This command saves the current rule setting file if it is modified. The command is disabled if you did not edit the file.

Save As

This command saves the current rule setting with the name specified in a *Save As* form.

Export Rules to CSV File

This command opens the *Export Rules to CSV File* form where you can peruse the directory structure and specify a file to save the rule set in a CSV format.

Export Rules

This command is used to export current rules into ASCII file. It will invoke a box like following.

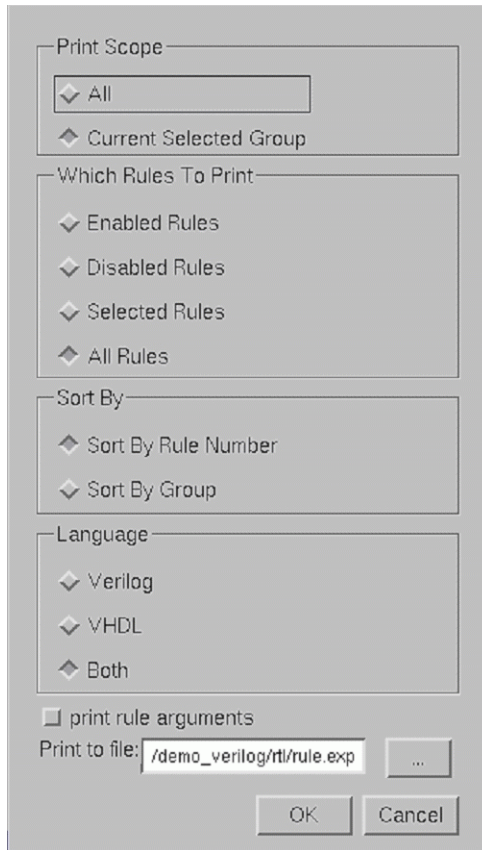


Figure: Export Rules

In above figure, in **Print Scope**, you can choose to print **All** rules supported by nLint or the rules under **Current Selected Group**. If you choose **All**, all the rules will be printed out flatten whether the rule is included in current group or not. If you choose **Current Selected Group**, you can choose to print the **Enabled Rules**, **Disabled Rules**, **Selected Rules** or **All Rules** in the scope. You can specify sorting order when rules are exported. The rules can be sorted in **Sort By Rule Number** or **Sort By Group**.

You can choose export language: **Verilog**, **VHDL** or **Both**.

You can toggle on **print rule argument** to export current argument value of the rules.

You can specify the file name to export rules with **Print to file** browser box.

Properties

This command shows all loaded RS files in a *Properties* form, as illustrated below.

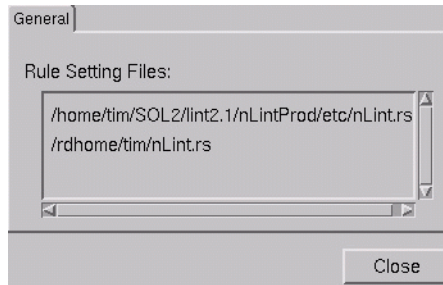


Figure: Show Loaded RS Files in Properties Form

Recent Files

This command displays the names of the most recently used files.

Close

This command closes the Rule Organizer. If you modified the settings and did not save the changes before leaving the Rule Organizer, nLint will open a Question dialog box to ask if you wish to save the changes or leave the settings unchanged.

If there is more than one RS file modified under the Administration mode, nLint will verify the request one by one. Under the Lint mode, if the merged RS file is not given a filename, it will be treated as a temporary file and will be discarded without confirmation. Notice that you may switch back to the Lint mode before linting your design. When you close the Rule Organizer, a Question dialog box appears, as shown below, verifying the request to switch the Administration mode to the Lint mode.

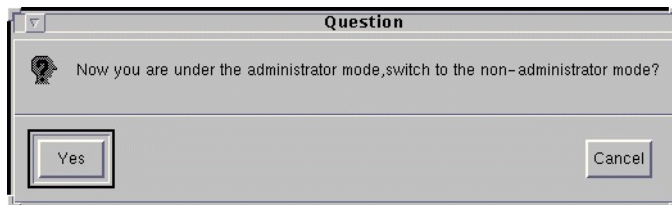




Figure: Question Dialog Box - Switching back from Administration Mode

Group Commands

Enable

Toolbar Button:



This command allows you to enable/disable the checking of current rule group. You can use the toggle button from the toolbar, or select the command from the menu bar (**Group -> Enable**). The group that is enabled is marked by the  icon; and the group that is disabled is marked by the  icon.

Non-configurable

Toolbar Button:



Only available under Administration mode. This command allows you to toggle on/off the status configuration of current rule group. You may use the toggle button from the toolbar, or select the command from the menu bar (**Group -> Non-configurable**). The non-configurable group is specified with green color, and the configurable group is shown as usual.

Rename

This command allows you to rename the current selected group. You can directly type the name of the group in the tree. To cancel the typing, press **Esc**.

New Top Group

This command allows you create a new top group with a default name, *NewTopGroupX*, where *X* can be 0, 1, ... You can change the name by using the **Rename** command.

Set As Active Top Group

This command allows you to set the current selected group as an active top group. nLint will check the rules under active group when linting.

Add Sub Group

This command allows you to add a sub-group under the current selected top group. If the current selected group is not a top group, this command will be disabled. Note that the new added group is named as *NewSubGroupX*, where *X* can be 0, 1, ... You can use the **Rename** command to change sub-group name.

Copy

After select the desired rule(s), you can invoke this command to copy the selected rule(s) to the specified sub group.

Paste

After a sub group is selected, this command pastes the rule(s) you have copied in the **Copy** command to a specified sub group folder.

Delete

This command deletes the selected group, no matter the top group or the sub-group.

Rules Commands

Copy

This command allows you to copy the current selected group. If the current top group is selected, the command is disabled.

Paste

This command allows you to paste the group copied by the previous command into the current selected sub-group. Before pasting, the rules should be checked to keep the rule number unique. If the current selected group is not a sub-group, it is disabled. If there is no suitable content in the clipboard, it is disabled.

Delete

This command allows you to delete the current selected rules. If the current selected group is top group, it is disabled.

Properties

Toolbar Button:



This command opens the *Properties* form where the properties of the selected rule are displayed. The properties include the rule name, location, rule number, rule attribute and arguments. You may configure the rule properties with the **Rules -> Properties** command.

NOTE: To view and configure the rule properties, you can double-click the rule, click the toolbar icon, or invoke the **Rules -> Properties** command from the menu. Changes of rule properties in this form are synchronized to the rule list in the *Rule Organizer* window.

Figure: Rule Properties Form

The *Properties* form shows the general information of the rule, such as rule name, location, rule number, its belonging language, its sub-language, index name and rule description.

- **Index Name**, which will be shown and can be reported in rule violation, is an alias to the rule number.
- **Description**, which can be shown in the list of rules, is similar to the rule name. You can customize these two fields.

NOTE: In the **Index Name** column of the *Rule Organizer* window, rules supporting SystemVerilog are shown in green; rules supporting Verilog but except SystemVerilog are shown in cyan and rules supporting VHDL are shown in purple.

- **Argument non-configurable:** When this option is enabled, configuration settings on the rule argument is not allowed. This option is similar to the

Rule -> Argument non-configurable command, it is only available under Administration mode.

- **Suppressed by translate off pragma:** This option is similar to the **Rule -> Translate Off** command.
- **Argument1:** This field sets the first argument for the rule. If there is no argument for the rule, this field will be grayed out.
- **Argument2:** This field sets the second argument for the rule. This field will be grayed out if the second argument is not available.

NOTE: Some rules only have the first argument, such as 21001; some rules have both the first and second arguments, such as 21013. Some arguments are set in selection fields; some arguments are specified by clicking the **Set Value** button to open the *Set Rule Argument Value* form where you can

set multiple arguments. Below is a list of argument types.

Argument Type	Rule	How to Set Argument
STRING	21047	Enter a value in the text field.
INT	21045	Enter a value in the text field.
ENUM	21001	Specify CASE_LOWER or CASE_UPPER from the selection field.
MULTISELECTED	27131	Click the Set Value button to specify from BLOCKING and NONBLOCKING check boxes.
SEQUENCE	22035	Click the Set Value button to set the order of CLOCK, RESET, SET, ENABLE, TE and CONTROL.
1. BOOL 2. SEQUENCE	22033	1. Specify TRUE or FALSE from the selection field. 2. Click the Set Value button to set the order of INPUT, OUTPUT, INOUT, BUFFER and LINKAGE.
LIST	24015	Click the Set Value button to write a list of argument. ";" is to separate the list; "," is to separate the items in the same list. You can use the Add , Modify , Remove and Remove All buttons for editing the lists.
MULTISELECTEDLIST	21022	Click the Set Value button to specify from RESET_LOW, RESET_HIGH; SET_LOW, SET_HIGH; LATCH_ENABLE_LOW, LATCH_ENABLE_HIGH, TRI_ENABLE_LOW, TRI_ENABLE_HIGH check boxes.

-
- **Severity:** This field shows the severity of the rule. You can change the severity by clicking the selection field.
 - **Status non-configurable:** If this option is enabled, configuration settings on the rule status is not allowed. This option is similar to the **Rule -> Status non-configurable** command, it is only available under Administration mode.
 - **Enable:** When a rule is checked (indicated by) in the Rule Organizer, the Attribute box in the *Properties* form will be enabled

(indicated by Enable). If you disable the Attribute box (indicated by Enable) and click **Apply**, the rule will be unchecked (indicated by).

- **Apply To All Languages:** This option applies the changes of rule properties to both rules for Verilog and VHDL languages. If this option is not enabled, rule changes will be applied to the current language only.
- **Apply To All Groups While Enable/Disable:** This option applies the changes of rule status (Enable/Disable) to all groups that contain the rule. You can disable this option if you want to make the changes on the current group only.

Map Severity

This command brings up a *Map Severity* form as follows.

Original Severity	Target Severity
Level1:Information	Level3:Error

Figure: Map Severity Form

By selecting `Level 1:Information` from the **Original Severity** combo box and `Level3:Error` from the **Target Severity** combo box, you may click **Add** to add this pair of severities into the list box. You can click **Remove** to delete a selected item in the list box and click **Remove All** to delete of them at once. When you click the **Apply** button, the severity level of all rules will be mapped. Clicking the **Cancel** button will close the form. Click the **OK** button will apply the above settings and close the form.

This function is useful for RS file backward compatible. In the earlier version of RS file, the severity of a rule is represented by a string. Any comparison is based on the severity string. User can insert any string as the severity of a rule. It is not so convenient to give an order of all rules by severity. In the latest version of RS file, the severity of a rule is represented by an integer, which is named `Level<n>`.

To be compatible, we give each severity level a string. Thus user can choose the severity level in a combo box for each rule.

Find Rules

Toolbar Button:



This command finds rules based on the settings for rules you are looking for. After a rule group in the left pane of the *Rule Organizer* window is selected, invoke this command to open the *Find Rules* form. After settings are specified, click the **Find** button in the *Find Rules* form, rules meeting your specification will be shown in the right pane of the *Rule Organizer* window.

You can copy-and-paste rules of your interest to a specified sub group on the left pane of the *Rule Organizer* window. Please refer to the [Group -> Copy](#) and [Group -> Paste](#) commands for more information.

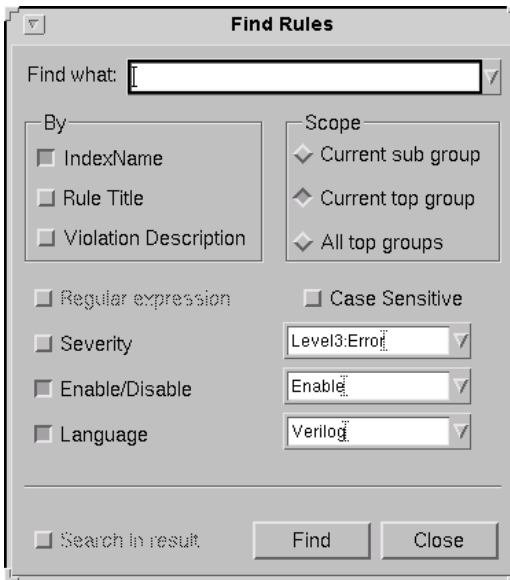


Figure: Find Rules Form

In the **Find what** text field, you can type a key word to search for rules.

In the **By** group, you can search according to index name, description and reference. If you specify **IndexName**, nLint will search rules by **Index Name** (Please see the red highlight in the first figure below). If you specify **Rule Title**, nLint will search rules by rule title (Please see the green highlight in the first

figure below). If you specify **Violation Description**, rules will be searched within the rule description of nLint rule documentation (Please see the red highlight in the second figure below).

IndexName	Description	Severity	Argument1	Argument2
27821	Incomplete assignment in always_comb	Level2		
27805	Task in 'always_comb' Block	Level2		
27662	dual set or reset detected	Level2		
27661	asynchronous reference in edge-sensitive	Level2		

Figure: Find Rules by Index Name and Description

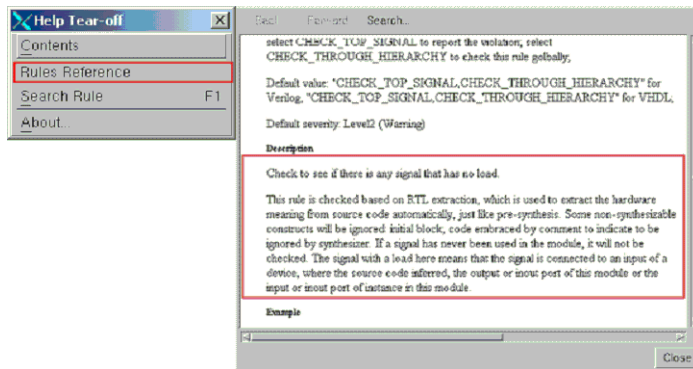


Figure: Find Rules by Reference

In the **Scope** group, you can specify the scope that you want to search for rules in the left pane of the Rule Organizer window. You can choose from one of the **Current Sub Group**, **Current Top Group** and **All Top Groups** options.

Regular Expression: If this option is enabled, the input string will be treated as a regular expression.

Severity: You can specify a desired rule severity.

Enable/Disable: This option allows nLint to find enabled or disabled rules.

Language: You can specify the language from Verilog or VHDL.

Search in result: If this option is enabled, nLint will search within the last found results. If this option is disabled, nLint will search within all nLint rules.

Click the **Find** button to search for rules meeting your specification.

To close the *Find Rules* form, click the **Close** button.

Select All

This command allows you to select all rules in the current rule group at one time.

Invert Selection

This command allows you to select the unselected rules in the current rule group once at a time.

Enable/Disable

Status Button: Enable: ; Disable:

This command allows you to enable or disable the checking for the current rule one by one. This is a toggle switch, which means that if the current rule is enabled, clicking this command will disable the rule. On the other hand, if the current rule is disabled already, this click action will enable it. You may also use the checkbox in the E/D column of rule list window under Rule Organizer.

Translate Off/On

Status Button: Translate Off: ; Translate On:

This command allows you to set the current rule to be suppressed by the Synopsys pragma `//synopsys translate_off` one by one. The pragma will suppress the synthesizable or non-synthesizable rules for coding style checking. This is a toggle switch, which means that if the current rule is suppressed by the pragma, clicking this command will disable the suppression.

Status non-configurable

Toolbar Button: 

This command is only available under Administration mode, it allows you to define the configuration settings on the rule status. This command is similar to the Status non-configurable option in the *Properties* form. The rule with non-configurable status is shown with gray color in the rule list window of Rule Organizer, and the one with configurable status is shown as usual.

Argument non-configurable

Toolbar Button:



This command is only available under Administration mode, it allows you to define the configuration settings on the rule argument. This command is similar to the **Argument non-configurable** option in the *Properties* form. The rule with non-configurable argument is shown with grey color in the rule list window of Rule Organizer, and the one with configurable argument is shown as usual.

View Commands

Sort

The rules can be sorted by three methods: either **By Check**, **By Number** (default), or **By Name**. When sorting **By Check**, the rules are sorted by the enabling status, i.e., rules that is disabled will be sorted first; rules that is enabled will be sorted next. When sorting **By Number**, nLint ignores all checks and characters except numbers. When sorting **By Name**, the checking rules are sorted by rule name alphabetically. When sorting by **Index Name**, the checking rules are sorted by index name alphabetically. In sorting by **Severity**, the checking rules are sorted by severity level.

NOTE: By default, nLint sorts checking rules numerically in ascending order.

Ascending

This command specifies that rules shown in the Rule Organizer are sorted in ascending order (A to Z or zero to 9).

Descending

This command specifies that rules shown in the Rule Organizer are sorted in descending order (Z to A or 9 to zero).

Language

Select to display the rules for **Verilog** or **VHDL**. You can choose **Both** for displaying all rules for these two languages. You can choose **Design** for displaying all rules according to the design. If the design is a mix-design, the rule organizer will decide the language by the top module of the design.

Administration Mode/Lint Mode

This is a toggle switch, select this command to display the working mode for **Lint Mode** or **Administration Mode**. The icons on the toolbar are changed when you change the working mode in Rule Organizer.

While switching from **Administration Mode to Lint Mode**, all RS files in Administration mode will be reloaded from disk in order to generate a merged RS file for design linting. A Question dialog box will appear verifying the request to save the modification of RS files one by one. If the modification is not saved, it will not take effect in the merged RS file.

While switching from **Lint Mode to Administration Mode**, any modification to the merged RS file will be discarded.

Preferences

This command is the same as the **Tools -> Preferences** command in the nLint Project Window.

Window Panel

This panel lists all active windows and enables you to switch among those windows.

Help Commands

Contents

This command brings up a Help Window for the whole tool. You can get help from the Contents, Index or Find tab.

Rules Reference

This command brings up a Help Window for Rule Category. You can get help from the **Contents**, **Index** or **Find** tab.

Search Rule

Bind Key: F1

This command provides a Help Window for you to look up the rule that you wish to find. When it is clicked, a help form with the rule you selected will be shown.

About

This choice offers information about the current nLint release, such as the current release's version and date.

Report Viewer

Overview

The Report Viewer appears after the nLint checking process is complete. This viewer consists of two sub-windows: Tree and Detail Windows. Tree window, at the upper part of the review viewer, shows the reported violations in a tree structure and sort them first by groups and then by rules. You can click on the plus (+) icon to expand the rule, thus showing more details, or click on the minus (-) icon to collapse, thus hiding the details.

When left-clicking on a violation, the detailed information will be shown in a detail window, at the lower part of the report viewer, with three sections: the first section is the violation itself; the second is the source code in quick view, enclosed with a rectangular box; and the third section (See Also) lists the related violations.

The reported violations are at the *leaf* level of the tree structure. When you double-click a violation, nLint will display the relevant source code in the nLint editor, or in the *nTrace/nSchema* window, depending on the current preference setting or violation.

There are two ways to invoke nLint Report Viewer to perform rule checking process: from main frame window (GUI mode) and from shell prompt (batch mode).

Lint from Main Frame Window (GUI Mode)

The **Run -> Lint** command in nLint main frame window starts nLint checker for rule checking. After the process is complete, a Report Viewer appears.

The tree window, at the upper half of the report window, lists the violations reported by nLint. You can expand or collapse the nodes in the tree to view or hide the report messages.

The detail window, at the lower half of the report viewer, shows detailed information of the violation. You can also view the details in a quick view mode, enclosed by a rectangular box, when an error is selected. This information includes the violated objects and the source code for the objects. You can show

or hide the source code in the rectangular box by clicking the plus or minus icon at the beginning of the text line.

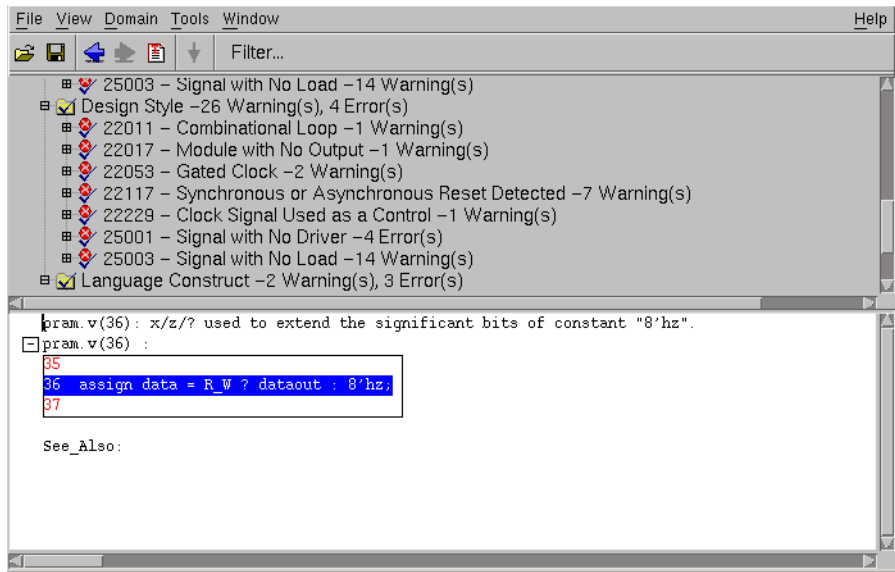


Figure: Report Viewer

Related violations with the same key object are listed in the *See Also* section for reference. In this figure, all violations violated on the object *data* are listed in this section. In addition, the related violations provide hypertext link to the corresponding violations in the tree window. You may click on the target violation and jump directly to the relevant violation in the report viewer.

After nLint checking process is done, use the **File -> Save** command in the report viewer to save the error result in a report DB file, if modified. The **File -> Open** command in the report viewer can be used to open an existing report DB file and view the saved error result.

Lint from the Shell Prompt (Batch Mode)

nLint Report Viewer can also be invoked from the command line as a batch run. When running in batch mode, nLint supports a command line option `-rdb`, which allows user to specify a file name for nLint to save the error result. The saved error report DB can be viewed by using the **File -> Open** command in the report viewer.

For the menu commands in Report Viewer, refer to the following sections.

File Commands

Open

Toolbar Button:



Use this command to open a report DB. After you choose a report DB directory, the report viewer will load the checking information from the report DB and show the information in the tree window of report viewer.

Save

Toolbar Button:



This command allows you to save the modified report DB file.

Save As

Use this command to save the report DB file with another filename.

Close

This command closes the report DB file.

Export Violations

Use this command to export the errors that are currently shown in the tree window of the report viewer into an ASCII file. In the output file, the errors are sorted with the same order in the report viewer.

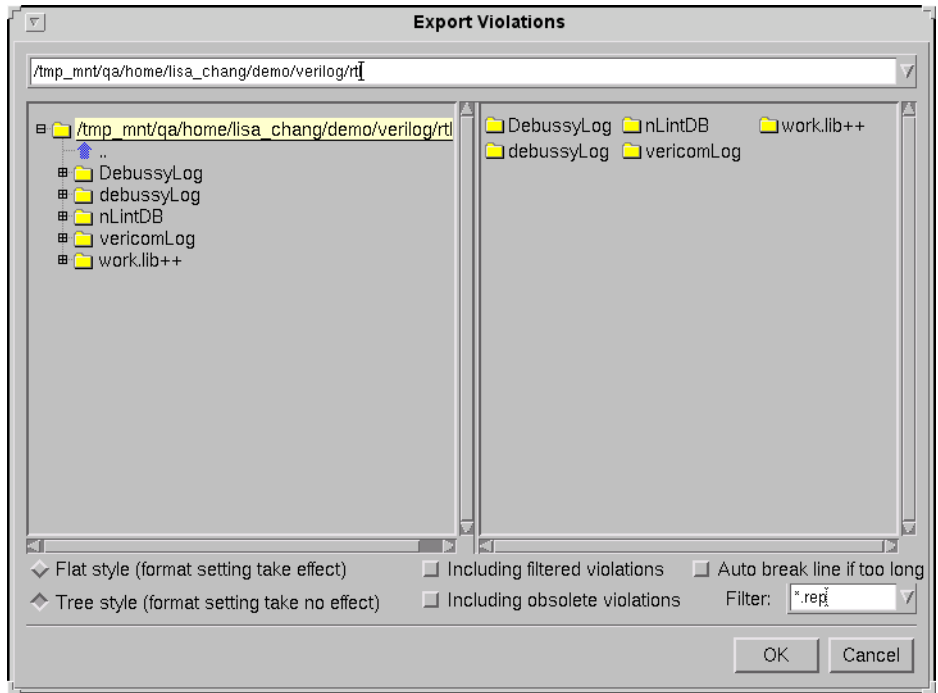


Figure: Export Violations Form

You can choose **Flat style** or **Tree style** to export the violations. If **Flat style** is enabled, the violations are exported based on the order of file and line. The output message format can be specified in the **Lint -> Message** tab of the **Tools -> Preferences** command and the output message format will take effect. And the **Auto break line if too long** option is disabled. If **Tree style** is enabled, the violations are exported just like they are organized in the tree view of report viewer. And the format setting will take no effect. If a line is too long, it will be broken down automatically if the option **Auto break line if too long** is toggled on.

When the **including filtered violations** option is enabled, violations including those filtered-out violations, will be exported to the specified report file. When the **including obsolete violations** option is enabled, violations, including those obsolete violations, will be exported to the specified report file.

Export Clock Domains

Use this command to export the clock domain information into a specified file. The format and contents in the file is similar to the one in the clock domain tree window of the report viewer.

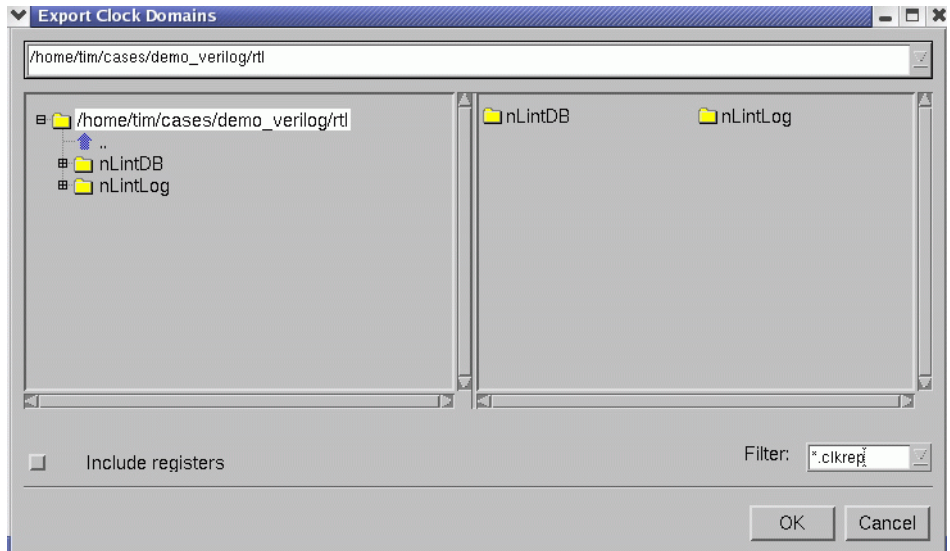


Figure: Export clock domains dialog

The **Include registers** check box allows you to export the registers of the clock domain.

Before use this command, the clock domain should be available. You can use **Run -> Create Domain** for clock domain extraction.

Close Window

This command closes the report viewer.

View Commands

Sort By

Use one of the following commands to select a sorting order for the error/violation messages.

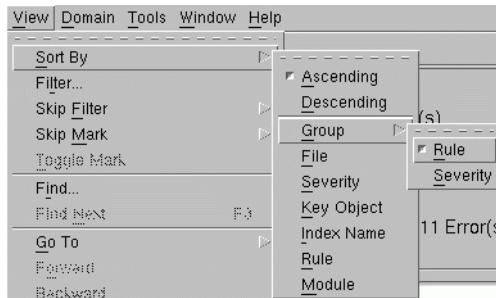


Figure: View -> Sort By -> Group

Ascending

This command allows you to sort the error messages by ascending order.

Descending

This command allows you to sort the error messages by descending order.

Group

You can choose to sort by rule groups -> rule number or rule severity.

Rule

This command allows you to sort the error messages or warnings by rule number in each group. nLint will assign a unique number for each rule, i.e., 22011 indicates the rule *combinational loop*. This command is used only when you view and process all errors of the same kind.

Severity

This command allows you to sort the error messages or warnings by severity in each group. For each rule in nLint, it can have different levels of severities. It is useful if you concern on the rule violations with high severity.

File

This command allows you to sort error messages or warnings by file name and line number. Once nLint has found any violation in the source files, nLint reports violated source file name and source line number. Select this command if you want to view and correct all errors file by file.

Severity

You can set severity level for each rule in Rule Organizer by **Tool> Rule Organizer> Map Severity**. For example, you can set the severity of some rules to Level2 with string **Error** and some to Level3 with string **Warning**. This command allows you to sort the error messages or warnings by severity in ascending or descending order in the first level.

Key Object

In general, nLint chooses a key object for most violations, i.e. *clock signal prefix/suffix checking*, the clock signal violated the rule is treated as the key object of the violation. In this way, you can group the related violation together by ascending or descending order. Thus you can have more information when you want to correct the error.

Index Name

This command allows you to sort the error messages or warnings by rule index name in each group. nLint allows user to configure an index name for each rule. Different rules can have same index name.


Rule

This command allows you to sort the error messages or warnings by rule number in the first level without rule group organization. nLint will assign a unique number for each rule, i.e., 22011 indicates the rule *combinational loop*. This command is used only when you view and process all errors of the same kind.

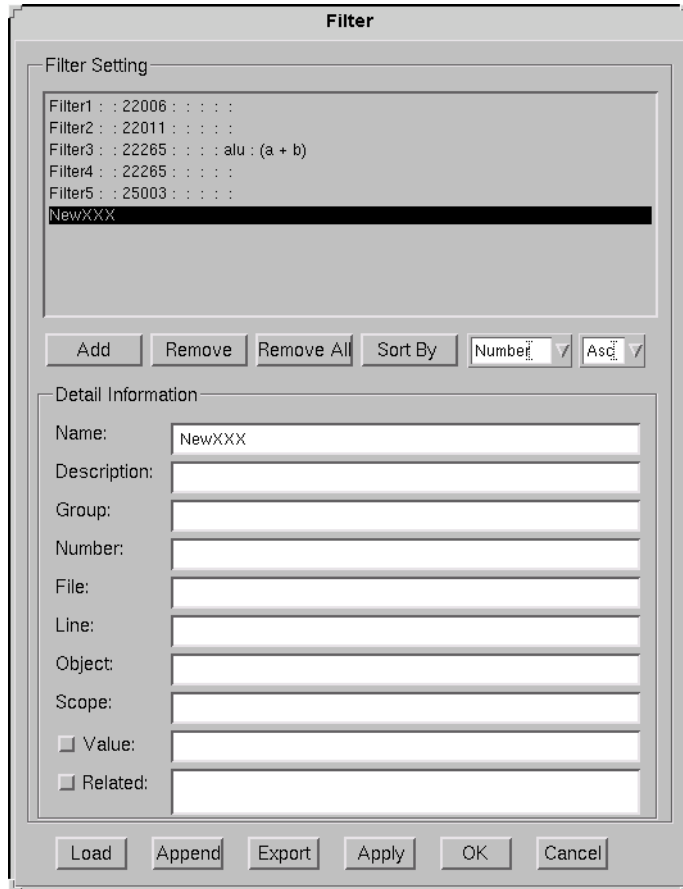
Module

This command sorts violation by module. In each module, violations are grouped by their severity. Violations that do not have module information will be put in a group named "null". The option "-sort m" to sort by module in batch mode is supported.

Filter

Toolbar Button: 

This command opens the *Filter* form where you can filter violation rules. The *Filter* form includes two sections: **Filter Setting** and **Detail Information**.



Filter	Number	Rule
Filter1	22006	
Filter2	22011	
Filter3	22265	alu : (a + b)
Filter4	22265	
Filter5	25003	
NewXXX		

Figure: Filter Form

- **Filter Setting**

The table in the **Filter Setting** section shows a list of filter with details corresponding to your settings in the **Detail Information** section.

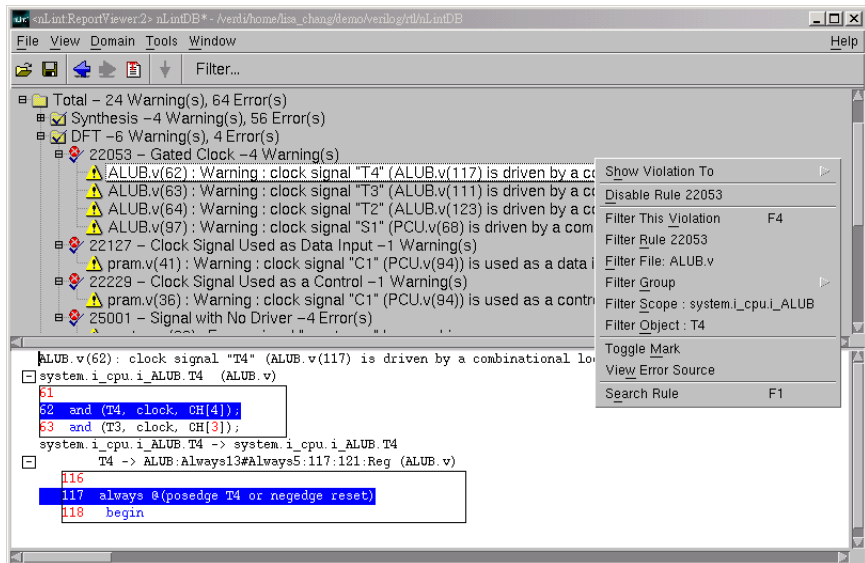
To create your desired filter, in the **Name** text field of the **Detail Information** section, replace the default name, *NewXXX*, with your preferred filter name by

entering a string. Text fields in the **Detail Information** section can be specified at the same time. For example,

```
Number = 21001, 21003  
File = PCU.v
```

It means to *filter rule 21001 and 21003 in file PCU.v*. After specifying other fields in the **Detail Information** section, click the **Add** option to complete the filter setting.

Alternatively, you can select a violation in the *Report Viewer* window and then click-right the violation to filter the violation.



To edit an existed filter, click the filter and then modify the fields in the **Detail Information** section. Any modification in the fields will be dynamically reflected in the table of **Filter Setting** section.

You can remove any filters by selecting the filter in the table of **Filter Setting** section and then clicking the **Remove** button. Clicking the **Remove All** button will remove all the filters.

You can use the **Sort By** option to display the filter(s) based on the **Number**, **Group**, **File** or **Object** fields. You can also use the **Asc** or **Dec** selection field to sort the filter in ascending or descending order.

- **Detail Information.**

This **Detail Information** section includes the following fields:

- **Name:** This text field shows the filter name. Note that period (.) is not allowed in this field.
- **Description:** This text field allows you to add comment for the selected violation.
- **Group:** This text field allows you to specify rule group of the violation. You can specify more than one group name in the field. Separate multiple entries with a comma.
- **Number:** This text field shows the rule number of the selected violation. You can specify more than one rule number in the field. Separate multiple entries with a comma.
- **File:** This text field shows the file name of the selected violation. You can specify more than one file in the field. Separate multiple entries with a comma.
- **Line:** This text field shows the line number of the violation.
- **Object:** This text field allows you to specify the key object in the violation. You can specify more than one object in the field. Separate multiple entries with a comma.
- **Scope:** This text field allows you to specify the scope that the violation is under. The scope can be a full hierarchy name or a module name. You can specify more than one scope in the box. Separate multiple entries with a comma.

The report viewer supports following three scope string:

1. **Full scope**, i.e. *system.i_cpu*, tells the report viewer to hide the violations under the specified scope. In this example, a violation with scope *system.i_cpu*, will be hidden, but the one with scope *system.i_cpu.i_ALUB* will not be hidden.
 2. **Full scope with a meta-character *** (*asterisk*), i.e. *system.i_cpu.**, tells the report viewer to hide the violations under the scope of its sub-scope. In this example, the violations with scope *system.i_cpu* or scope *system.i_cpu.i_ALUB* will be hidden.
 3. **Module**, i.e. *ALUB*, tells the report viewer to hide all violations with scope as the module. Those violations are checked in modules, which will be detected as violations in every instance in the design.
- **Value:** This text field is a specific field used in smart filtering “this violation.” It will be different by different rules. Generally, it is the value of arguments of the rule when checking.

- **Related:** This text field is similar to the Value field, which is also a specific field used in smart filtering “this violation.” It will be different by different rules. Generally, it is the path information of the violations.

The **Load** button will load the filter settings from a saved file to replace the current settings.

The **Append** button will load the saved file and append the filter settings to the current settings.

The **Export** button will save all the filters to a file as following:

```
@nLint rc file Version 1.0
[Violation3]
Number = 22056
[Violation2]
File = ALUB.v
[Violation1]
Number = 22011,22013
File = PCU.v
```

The **Apply** button will apply the filters on all violations in the report viewer.

The **OK** button will apply the filter on all the violations in the report viewer and close the *Filter* form.

The **Cancel** button will discard all the changes and close the *Filter* form.

Skip Filter

This command allows you to further configure the filtering process by using the following sub-commands.

Skip None

Skip nothing.

Skip Filtered

This command allows you NOT to show the filtered violations in the report viewer.

Skip Non-filtered

This command allows you NOT to show the un-filtered violations in the report viewer.

Skip Mark

This command allows you to further configure the marking process by using the following sub-commands.

Skip None

Skip nothing.

Skip Marked

This command allows you NOT to show the marked violations in the report viewer.

Skip Non-marked

This command allows you NOT to show the un-marked violations in the report viewer.

Toggle Mark

This command allows you to toggle the "read mark" on messages. When you double-click on an error message, the corresponding source code will be shown in the built-in editor or in *nTrace* window. At the same time, the color of that error message line will also be changed, in order to indicate that the line has been viewed and processed. The change of the line color indicates that a "read mark" has been placed on that message line.

Alternatively, you can select a message line, which has no "read mark" on it, and issue the **Toggle Mark** command. A "read mark" will then be placed. Notice that if the selected line already has a "read mark" on it, this command will remove the mark.

This command can also work at the rule level when the error messages are sorted by rule number. By selecting a rule and issuing this command, the read marks on all messages under that rule will be toggled.

Find

This command allows you to find violation by rule number, name string or key object name. When you invoke this command, a *Find Violation* form will open.

Find what: IDB

By rule number

By string

By object

Direction

Up

Down

Find Close

Figure: Find Violation Form

Type any text string in the **Find What** text field, and specify the search direction and criteria; then press **Enter** or click **Find** to start searching the violation. The search result shows the first violation that matches your specification. To close the form, click **Close**.

Find Next

Toolbar Button:



This command allows you to move current selected violation to the next satisfied violation. When you find a certain violation by the **View -> Find** command, you can use this to go through all satisfied violations. Or, when dragging a signal from the Verdi system to locate the violations, you can also use this command to go through all satisfied violations.

Go To

Use this command to move your attention to one of the following violations.

Next Violation

Move forward to the next violation and select it.

Prev Violation

Move back to the previous violation and select it.

First Violation

Jump to the first violation and select it.

Last Violation

Jump to the last violation and select it.

Forward

Toolbar Button:



This command allows you to move forward in the history list of the once selected violations.

Backward

Toolbar Button:



This command allows you to move backward in the history list of the once selected violations.

Add See Also

This command appends hypertext link to the relevant violation(s) in the detail window. This is a toggle switch. When it is on, the violations with the same key object of the selected violation will be appended with reference link to the corresponding violations.

Domain Commands

Specify It As Source

This command allows you to specify the selected root of the clock source to be a clock source. If the root of the clock source is not resolved by some reason, you can use this command to specify it as a clock source.

Find Register

Use this command to specify the name of register signal and find it out in the tree window of clock domain.

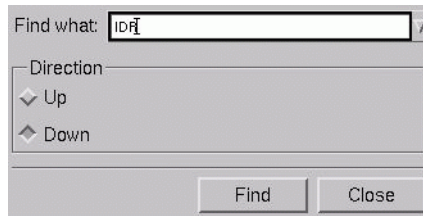


Figure: Find Register Dialog

Find String

This command serves a similar function as Find Register. It facilitates finding desired string in clock source tree. If the string is found, select the item. You can click the **Find** button again to search for the next string. If a string can not be found, a warning of "Cannot find xxx" clock analysis appears.

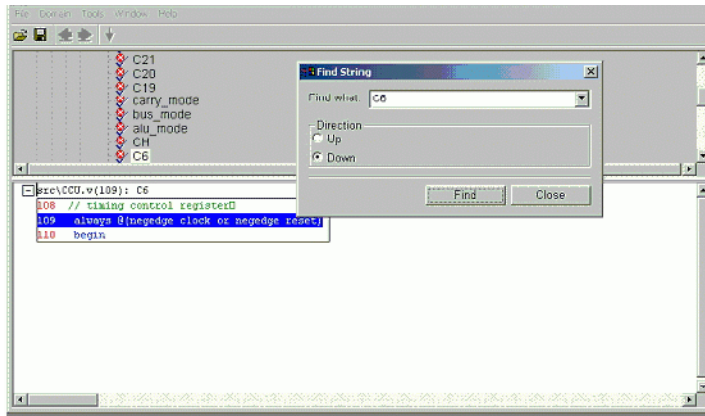


Figure: Find String

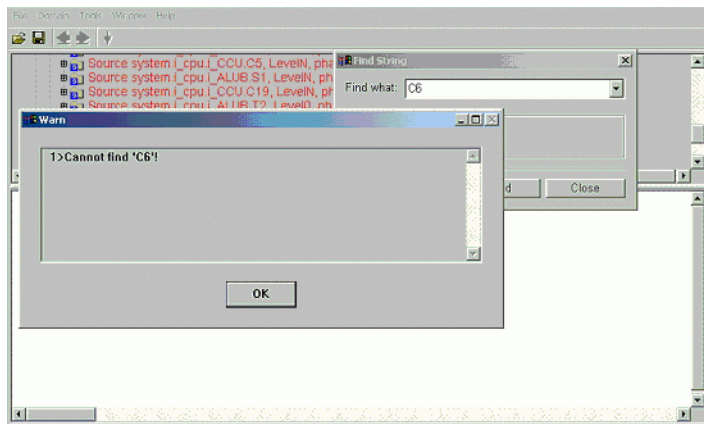


Figure: Cannot find 'xxx' string!"

Hide

Use this command to hide the paths or the registers in the tree window of clock domain. It is useful if the path between clock source signals to the clock signal is too long or there are many registers listed for a clock signal.

The following figure shows the report view after some contents are hidden.

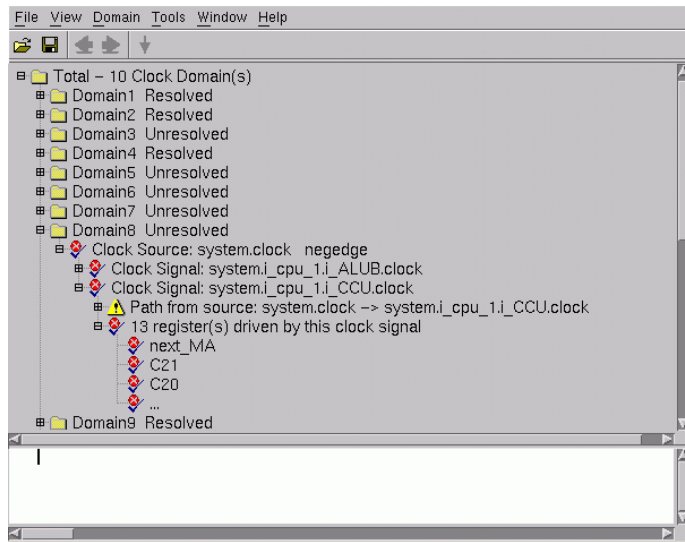


Figure: Some Registers are Hidden

When right-clicking on the "...", the **Expand** command appears for you to expand the contents.

Expand

When something hidden in the tree window of clock domain, this command allows to expand them out.

The following figure shows the report view after some contents are expanded.

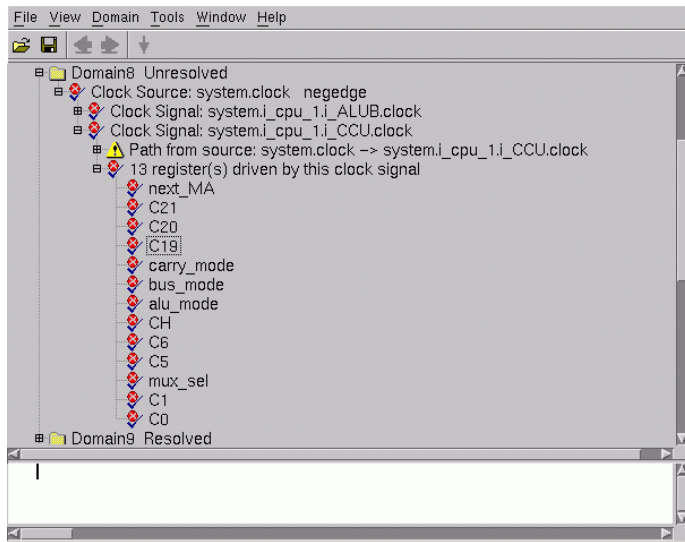


Figure: All Registers are Expanded

When right-clicking on the register or path, the **Hide** command appears for you to hide the contents.

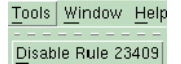
Move To

This command provides four sub-menu - Next Domain, Prev Domain, First Domain and Last Domain, which allows you to move quickly to the next, previous, first or last domain.

Tools Commands

Disable Rule

This command is similar to the **Rules -> Disable** command in Rule Organizer. After you have reviewed an error message, if your focus is not on that error type, you can use this command to disable that rule, i.e., *Disable Rule 22081*. **Disable Rule number** is displayed as the right figure



When you click on a violation of which the rule is disabled previously, the command will become **Enable Rule number** for you to enable the rule.

Without invoking the Rule Organizer, this command provides another direct and convenient choice for configuration.

View Error Source

This command shows the relevant source code to the current error message with the tool defined in the **Tools -> Preferences** command. This command is equivalent to the double-click action on the current error message.

Preferences

This command is similar to the **Tools -> Preferences** command in the Project Window of nLint main frame window.

Commands in Shortcut Menu

The report viewer also provides a quick access to the commonly used commands in a shortcut menu. Different commands appear in the shortcut menu when selecting a rule or violation.

Violation

When right-clicking on a violation, a shortcut menu appears, as shown below.

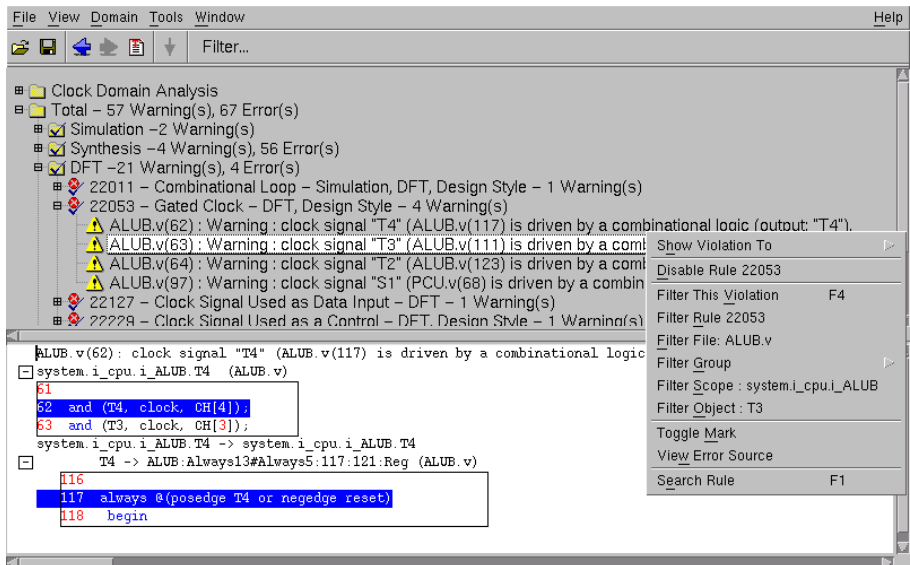


Figure: Shortcut Menu Appears when Clicking on Violation

The shortcut menu includes

Show Violation To

This command provides three sub-commands, **Default Editor**, **nTrace**, and **nSchema**, for you to show the violation in the preferred display window.

Disable Rule

Please refer to the **View -> Disable Rule** command for more details.

Filter This Violation

Bind Key: F4

This command filters the specified violation and record the key information of this violation in the *Filter* form as a filter item.

Take *21001, signal name case*, as an example, the information of scope and key object name will be recorded. If there are changes such as adding lines to the file, the filter settings still take effect. Key information aims to distinguish current violation with other violation, especially violations in the same rule. Note that key information is NOT sensitive to the modification of code such as changing line number.

Filter Rule

It is a fast way to filter the rule of the selected violation. The behavior is just the same as specifying the rule number in the *Filter* form and click **OK**.

Filter File

It is a fast way to filter the file of the selected violation. The behavior is just the same as specifying the file name in the *Filter* form and click **OK**.

Filter Group

It is a fast way to filter a certain rule group of the selected violation. The behavior is just the same as specifying the group name in the *Filter* form and click **OK**.

Filter Scope

It is a fast way to filter a scope of the selected violation. The behavior is just the same as specifying the scope/module name in the *Filter* form and click **OK**.

Filter Object

It is a fast way to filter the key object of the selected violation. The behavior is just the same as specifying the key object name in the *Filter* form and click **OK**.

Toggle Mark

Please refer to the **View -> Toggle Mark** command for details.

View Error Source

Please refer to the **View -> View Error Source** command for details.

Search Rule

Bind Key: F1

Please refer to the **Help -> Search Rule in Rule Organizer** command for details.

When right-clicking on a rule, only the **Disable Rule**, **Filter Rule** and **Search Rule** are available in the shortcut menu.

Windows Panel

This panel lists all active windows and enables you to switch among those windows.

Help Command

Search Rule

This option provides a Help Window to check rule of your interest. You can get help from **Project Window -> Help -> Contents**, **Index** or **Find**.

Pragmas

Overview

Pragmas are special comments embedded in the source code, which can be used to control the behavior of nLint. There are two types of pragmas recognized by nLint: (1) those starting with `//novas`, and (2) those starting with `//synopsys`.

//novas Pragmas

These special comments are specific to nLint. The syntax and semantics are listed below.

Syntax

```
<novas_progma> ::= //novas {(<line_number>)} <comment_item> {,  
<comment_item>}*  
<comment_item> ::= push  
                    | pop  
                    | {<confines>}[+|-] [<rule_number> | <group_name>  
| All]{:<line_number>}  
<line_number> ::= [<absolute_number> | <corresponding_number>]  
<absolute_number> ::= #<number>  
<corresponding_number> ::= <number>  
<confines> ::= [s | m | b]
```

Note that each `novas_progma` should take up a line. There should be no comments leading the pragma, and nothing except an optional `// comment` trailing the pragma. Pragmas cannot span lines. If you have a long list of items, you can break it up into multiple pragmas.

Semantics

- `+<rule_number>` means to enable checking for `<rule_number>`.

- `-<rule_number>` means to disable checking for `<rule_number>`.
- `+<group_name>` means to enable checking for all rules in `<group_name>`.
- `-<group_name>` means to disable checking for all rules in `<group_name>`.
- `+All` means to enable checking for all rules.
- `-All` means to disable checking for all rules.
- `push` means to save the current enabled/disabled status for all rules.
- `pop` means to restore the enabled/disabled status for all rules which have been saved since the last push.
- `:<line_number>` means the enable/disable action will not take effect until the line number is, absolutely or relatively, specified.
- `<confines>` means to show the range in which the pragma works.

s: means the pragma works at a single line.

m: means the pragma works in a module.

You have to put the pragmas above the module. Except blank lines and comment lines, nothing can be inserted between the pragma directive and the module.

b: means the pragma works in a block

A "block" means an initial block, an always block, a task or a function. You have to put the pragmas above the block statements. Except blank lines and comment lines, nothing can be inserted between the pragma directive and the block.

Usage Notes

- Nested push and pop are not supported. After a push is encountered, all subsequent push pragmas will be ignored until a pop is seen.
- If a pop is encountered without a corresponding push, it will be ignored.
- Novas pragmas take precedence over the rule setting file. However, they don't have any permanent effect on the contents of the rule setting file.
- When conflicting items are found in a pragma, latter ones will prevail.

For example:

```
//novas +Synthesis, -22075
```

This will enable all rules in Synthesis except for 22075.

```
//novas -22075, +Synthesis
```

This will enable all rules in Synthesis including 22075. (i.e., since 22075 is under Synthesis group, "-22075", in this example, is redundant.)

- When Novas pragmas with <confines>, m or b, are specified, the disabled/enabled rule checking will take effect in the defined confines. Below a module or a block, the effect vanishes.

For example:

```
line 21: ...
line 22: //novas m-22075
line 23: module test(°K);
...
line 40 endmodule
line 41 ...
```

When the pragma at line 22 is encountered, rule settings before line 22 will be registered. At line 22, rule 22075 is explicitly disabled for the succeeding module, which is from line 23 to line 40. From line 41, the effect of the pragma at line 22 vanishes and the registered rule setting is restored.

Examples

```
line 20: //novas push, -22001, +22003
line 21: ...
line 22: ...
line 23: ...
line 24: //novas pop
line 25: ...
```

At line 20, first the current rule setting is remembered, then the checking for rule 22001 is explicitly disabled while the checking for rule 22003 is explicitly enabled. At line 24, the rule setting is restored to the one at the beginning of line 20.

```
line 20: //novas -22001:2
line 21: ...
line 22: ...
line 23: ...
line 24: ...
line 25: ...
```

At line 20, the checking for rule 22001 is explicitly disabled for the following two lines, which is until line 22. This pragma is the same as "//novas(20) -22001

```
line 20: //novas -22001:#23
line 21: ...
line 22: ...
line 23: ...
line 24: ...
line 25: ...
```

At line 20, the checking for rule 22001 is explicitly enabled until line 23. This pragma is the same as `//novas(#23) +22001`

```
line 20: assign out = in; //novas s-22001, s-22003
```

At line 20, the checking for rules 22001 and 22003 are explicitly disabled.

```
line 20: module test(...);  
...  
line 30: //novas b-22001, b-22003  
line 31: always_latch begin  
...  
line 37: end  
line 38: ...  
line 50: endmodule
```

When the pragma at line 30 is encountered, rule settings before line 30 will be registered. At line 30, rule 22001 and 22003 are explicitly disabled for the succeeding block, which is from line 31 to line 37. From line 38, the effect of the pragma at line 30 disappears and the registered rule setting is restored.

```
line 20: // novas m-22001  
line 21: module m1(clk, hard_rst);  
...  
line 30: //novas +22001  
line 31: always @(posedge clk)  
line 32:     begin  
line 33:         last_ds2_gnt_r <= hard_rst ? 3'b001 :  
last_ds2_gnt_ns;  
line 34:     end  
line 35: endmodule  
line 36:
```

When the pragma at line 20 is encountered, rule settings before line 20 will be registered. At line 20, rule 22001 is explicitly disabled for the succeeding module. However, at line 30, the checking for rule 22001 is explicitly enabled again. Since the latter pragma has higher priority, from line 31 to line 35, rule 22001 is enabled. From line 36, the rule setting is restored to the registered setting before line 20.

//synopsys Pragma

nLint supports the following Synopsys compiler directives:

```
//synopsys translate_on  
//synopsys translate_off  
//synopsys full_case
```


`//synopsys translate_on` and `//synopsys translate_off` will influence the checking for those rules which are marked with "Translate_Off=ON" in the rule reference manual. As soon as a `//synopsys translate_off` is encountered, all those rules are temporarily disabled, and as soon as a `//synopsys translate_on` is encountered, the checking for those rules will automatically resume.

`//synopsys full_case` will influence the checking for rule 23007, which checks to see if a case statement is full or not. If `//synopsys full_case` is used, the violation report will be suppressed even if the case statement is not full.

Command Line Options

Command Line Syntax

The command line syntax to invoke nLint is shown below:

Usage:

```
nLint: {[GUI mode options] | [batch mode options]} [general options] [simulator options] [tcl debug options]...
```

[General Options]

-h	Print this usage.
-nologo	Suppress the display of nLint logo at startup.
-logdir logDirectory	Specify the location of the log directory.
-logfile logFile	Specify the location of the log directory/file.
-ssr session_file	Load a session file.
-rcFile rc_file	Specify the nLint resource file (.rc).
-opt filename	Use the command option file to write more options.
-verilog	Specify Verilog as the language type to import design from source.
-2001	Specify to accept selected Verilog 2001 syntax.
-sv	Specify to accept selected SystemVerilog syntax.
-ssy	Do not automatically tag library modules in library directory (-y) as library cells.
-ssv	Do not automatically tag library modules in library file (-v) as library cells.
-ssz	Ignore `celldefine compiler directives.
-top top_module	Specify a top module when importing the design.
-lib lib_name	Specify the library name.
-comment_transoff_regions	Ignore the code between "synopsys translate_off" and "synopsys translate_on".
-readonly <on off>	Specify whether to open the source file in read-only mode. The default is OFF.

-rdb DBdirectory	Print checking result to DB directory. Default is "nLintDB" under working directory.
-vf format_string	Specify the output message format.
-beauty	Beautify the output file.
-fullname	Show the full object name in output messages.
-detail	Show detail message.
-no_syntax	Ignore compiler errors.
-lineMapping on/off	Report on mapped file and line, just for verilog 2001. Default is OFF.
-filter <file>	Specify the filter setting file.
-clk_export_reg	Specify to print the register information when printing clock domain. By default, the register will not be printed out.
-rs rs_files	Specify rule setting files.
-active_group <top_group>	Specify the active top group by <top_group> for checking.
-nors install:home:current	Do not merge the specified rule setting files.
-drm	Do not merge rule setting files; only the last one will take effect under this option.
-er filename	Specify an output file to which the Rule Organizer's active top group enabled rules are written.
-sev_map levelX=severity_string	Specify string for severity level.
-lintTop entry_scope	Indicate the design scope that nLint should start the linting.
-sm module_name	Suppress the specified module.
-smr module_name	Suppress the specified module and its sub-modules recursively.
-bb module_name	Specify the modules to treat as macro cells.
-bs file_name	Specify ONE file that all modules in it are treated as macro cells.
-bf file_name	Specify a file with a source file name list in which all modules in the file are treated as macro cells.
-df file_name	Specify a source file that should not be checked.
-uf file_name	Specify a file with a source file name list that should not be checked.
-r rules	Specify the rules not to be checked; it will override Novas pragma in source file.

+r rules	Specify the rules to be checked; it will override Novas pragma in source file.
-pr rules	Specify the rules not to be checked; Novas pragma in source file will override this.
+pr rules	Specify the rules to be checked; Novas pragma in source file will override this.
-ignore_pragma	Ignore Novas pragma in source code.
-wn warn_num	Specify maximum number of warnings for this run.
-pwn warn_num	Specify maximum number of warnings per rule.
-maxseverity severity_str level_num	Specify maximum severity of warning to be reported.
-minseverity severity_str level_num	Specify minimum severity of warning to be reported.
-ignore_initial	Ignore linting in 'initial' blocks.
-lint_cell_lib	Turn ON to lint library files. The default is OFF.
-libcellas [hier macrocell comb]	Treat lib cell as hierarchy module or macro cell or complex comb. Default is hierarchy module.
-actualwidth	Ignore leading zero for constants without specified size when checking width rules. It is the default setting.
-actualwidth_off	Turn off -actualwidth.
-vs port_name=value	Specify the value to propagate.
-partition_top	Specify that only top module is the partition.
-partition_module	Specify every module is a partition.
-partition <scope>	Specify that the partitions are the instances under the scope specified.
-power_def symbol	Specify the pre-defined power symbol.
-ground_def symbol	Specify the pre-defined ground symbol.
-power_def_off	Turn off the pre-defined power symbol.
-ground_def_off	Turn off the pre-defined ground symbol.
-power_file symbolfile	Specify the user-defined power symbol-mapped file.
-ground_file symbolfile	Specify the user-defined ground symbol-mapped file.

- pass_through_assignment <on/off>	Specify whether to pass through assignments when checking DFT rules. The default is ON.
- pass_through_buffer <on/off>	Specify whether to pass through buffers when checking DFT rules. The default is ON. Please note that when -pass_through_buffer is on and rule 22055 is enabled, a prompt message will pop up: "-pass_through_buffer is 'on'. The engine will ignore all buffers when checking 22055. So no violations of 22055 will be reported."
- pass_through_inverter <on/off>	Specify whether to pass through inverters when checking DFT rules. The default is ON. Please note that when -pass_through_inverter is on and rule 22054 is enabled, a prompt message will pop up: "-pass_through_inverter is 'on'. The engine will ignore inverter pairs when checking 22054. So no violations of 22054 will be reported."
-scan_in_prefix <prefix>	Signal with specified prefix will be treated as scan in signal. nLint will ignore checking connection rules on such signals.
-scan_in_suffix <suffix>	Signal with specified suffix will be treated as scan in signal. nLint will ignore checking connection rules on such signals.
-scan_out_prefix <prefix>	Signal with specified prefix will be treated as scan out signal. nLint will ignore checking connection rules on such signals.
-scan_out_suffix <suffix>	Signal with specified suffix will be treated as scan out signal. nLint will ignore checking connection rules on such signals.
-scan_clock_prefix <prefix>	Signal with specified prefix will be treated as scan clock signal. nLint will ignore checking connection rules on such signals.
-scan_clock_suffix <suffix>	Signal with specified suffix will be treated as scan clock signal. nLint will ignore checking connection rules on such signals.
-we_mem memory_enable_name=value	Specify the memory W/E signal.
-rtl_level <n>	n is the level of detail RTL. By default, it is 10.
-rtl_muxlatch <on/off>	Turn on/off the mux style latch. The default is OFF.
-rtl_fsm <on/off>	Turn on/off FSM extraction. The default is ON.
-rtl_expand <on/off>	Turn on/off expanding GENERATE Block. The default is ON.
-ex_clk	Specify to extract clock domain.
-vclk_source <vclk source>	Specify a virtual clock source name.

-clk_source <signal>+VS_<name>	Specify a clock source signal or a clock source signal under a virtual clock source.
-gen_clk_source NM_name+m/ d<n>+p<n>+MA_m aster+PS_<pin/ signal>	Specify a generated clock source.
-clk_uncern <signal>	Set unconcern signal for clock domain extraction.
-uncern_cellport <cell fromPort:toPort>	Set unconcern cell port for clock domain extraction.
-uncern_instport <instance: fromPort: toPort>	Set unconcern instance port for clock domain extraction.
-gated_clk_cell <cell.port>	Specify the gated clock setting.
-use_polarity <on/ off>	Clock analysis with polarity information or not. The default is ON.
-pass_gated_clk <on/off>	Pass through gated clock source when extracting clock domain. The default is ON.
-ignore_cell <cell>	Ignore latch/register cells in clock path.
-clk_lasr <on/off>	Specify to treat latch storage as register in clock domain extraction or not. The default is ON.
-clk_masr <on/off>	Specify whether to treat memory storage as a register in clock domain extraction. The default is ON.
-udr udr_directory	Specify the user-defined rule directory.
-udronly	Specify to check user-defined rules only.
-noudr	Specify not to check user-defined rules.
- pass_through_GATE CLK <on/off>	Specify whether to pass through GATECLK when checking DFT rules. The default is ON.
-check_ parameterized_ modules <on/off>	Specify whether to check parameterized modules or not. The default is OFF.
-treat_latch_enable_ as_control <on/off>	Specify whether to treat "latch enable" as control signal or not. The default is OFF.

[GUI Mode Options]

-gui	Invoke with GUI mode.
-lint	Specify to lint the design when the GUI starts up.
-auto_compileon/off	Specify whether to do compilation automatically after the design is loaded. The default is ON.
-over_lint_ds	<p>Specify not to generate file "nLint.ds". By default, nLint will produce a file "nLint.ds" when quit nLint GUI. When nLint is invoked, if the command line includes -over_lint_ds, nLint will not output this file "nLint.ds" when quit GUI.</p> <p>Example:</p> <ol style="list-style-type: none"> 1. nLint -gui test.v -over_lint_ds. Option -over_lint_ds is included and nLint.ds will not be output. 2. nLint -gui test.v. Option -over_lint_ds is not included and nLint.ds will be output. <p>When nLint -opt nLint.ds is specified, nLint.ds includes original options. When you quit nLint, nLint will save these settings to nLint.ds. Below is the project setting options which can be saved to nLint.ds.</p> <p>-lintTop, -top, -we_mem, -bb, -bb_disable, -bs, -partition, -partition_disable, -partition_module, -partition_top, -vs, -vs_disable, -clk_uncern, -clk_uncern_disable, -uncern_cellport, -uncern_cellport_disable, -uncern_instport, -uncern_instport_disable, -gated_clk_cell, -clk_source, -clk_source_disable, -gen_clk_source, -gen_clk_source_disable, -clk_source, -clk_source_disable, -vclk_source, -vclk_source_disable, -sm, -sm_disable, -smr_disable, -df, -df_disable, -ignore_cell, -ignore_cell_disable, -scan_in_prefix, -scan_in_suffix, -scan_out_prefix, -scan_out_suffix, -scan_clock_prefix, -scan_clock_suffix, -sdc, -sdc_spt, -sdc_top, -cts, -cts_spt, -cts_top, -reset_gen_module, -reset_gen_module_disable, -clock_gen_module, -clock_gen_module_disable, -tvs, -tvs_disable, -tport, -tport_disable</p>

[Batch Mode Options]

-out output_file	Specify the file where nLint writes out the checking results. If the output file name equals "screen", nLint will print the violations on screen.
-clk_export <filename>	Specify a file name to print the clock domain out.

-on_rule_exit All <rule_number> AnyCompilerError	Indicate to stop linting if a violation is detected; use “All” to represent any rules; use “rule_number” to represent rule number; use “AnyCompilerError” to represent compiler error number.
-ee exit_value	Specify nLint's returned value when violations are found.
-ignore_latch_cell <latch>	Specify the latch cells or registers to ignore during clock extraction.
-reset_gen_module <module>	Specify the global reset generator modules.
-clock_gen_module <module>	Specify the global clock generator modules
-max_compile_error <Num>	Specify the maximum number of compilation error.
-sort f s r k	Specify the sorting order of violations, which are to be saved in the output file, by the following sequence: file (default), severity, rule, and key object.
-vericom [on/off]	Specify whether to compile the design to library. The default is OFF.
-local_lint	Lint the design incrementally.
- save_obsolete_violat ions [on/off]	Specify whether to save the obsolete violations in violation data base. The default is ON.
-out_filter [on/off]	Export the filtered violations to the output file.
-out_obsolete [on/off]	Export the obsolete violations to the output file.
-critical module_name/ instance_name	Specify module or instance to check rule 25014.

[Simulator Options]

Verilog simulator command line options (e.g., -f run.f, etc.)

[TCL Debug Options]

For details on UDR script debugging, refer to the readme file under the <nLint_install_dir>/etc/TclPro/ directory.

Option `-actualwidth/-actualwidth_off`

These two options are used to indicate the significant bit when calculating bit width on integer constant. When `-actualwidth` is used, nLint will consider the significant bit position in calculating bit width of constant (without size specified) in checking width rules. When `-actualwidth_off` is used, nLint will not consider the significant bit position in calculating bit width of constant (without size specified) in checking width rules. The default setting is `-actualwidth`.

Some examples, listed below, are provided to show the behavior of nLint.

`4'h1` : the width is 4 no matter `-actualwidth` or `-actualwidth_off`.

`'h1` : the width is 1 no matter `-actualwidth` or `-actualwidth_off`.

`1` : the width is 1 when `-actualwidth`; the width is 32 when `-actualwidth_off`.

`4'b1111 << 3` : the width is 4 no matter `-actualwidth` or `-actualwidth_off`.

`'h1 << 3` : the width is 1 no matter `-actualwidth` or `-actualwidth_off`.

`1 << 8` : the width is 9 when `-actualwidth`; the width is 32 when `-actualwidth_off`.

Option `-logdir`

This option indicates where nLint puts the log file. By default, if there is a directory named LOG in user's home dir, nLint will set a random log dir like `nLint_3890` under LOG, otherwise the log-dir will be located in the work dir.

For example, nLint `ALUB.v -logdir AAA`, then the `nLint.rc/compile.log/turbo.log` will be put in the dir AAA.

Option `-logfile`

This option specifies the logfile name. By default, it is `turbo.log` under log dir.

Option `-top top_module`

For the rules nLint checks, some are module-based rules such as *incomplete sensitive lists*; the others are global rules such as *combinational rules*. The global rules are related to a path with scope. They can be checked under a determined scope. When there is more than one top level module in the design to be linted, nLint will choose the first one to check those global rules.

This option is used to indicate which top module is the one for nLint to elaborate the design. Other top level modules are ignored by nLint.

If the design has more than one top module and `-top` is not specified, nLint will print a warning message and choose the first module as the top module.

If `-top <top_module>` is specified but not in the design, nLint will also print a warning message and choose the first module.

Option `-lintTop entry_scope`

Assumed that the design is built by a team of designers. After the design is integrated, if one of the designers would still like to lint his/her sub-design but in the integrated environment, this option is useful. Considering the design file containing test-bench part, with this option you do not need to write a special design file for nLint to lint the design part. The `entry_scope` can be a module of the top one of the sub-design. It indicates nLint to begin to lint only the sub-design specified by `entry_scope`.

This option is different with `-top top_module`. Considering there are parameters overriding, lint from a sub-design is different with treat the sub-design as top one.

Enable/Disable a rule in command line `-r/+r`

This option is used to enable or disable a rule in command line. It is useful to quickly test a rule of nLint on a case. For example,

```
-r All +r 21001
```

This setting will override pragma embedded in source code.

Enable/Disable a rule in command line **-pr/+pr**

This option is used to enable or disable a rule in command line. It is useful to quickly test a rule of nLint on a case. For example,

```
-pr All +pr 21001
```

the above command line will disable all rules but enable only 21001.

The different to `-r/+r` option is that, this setting will be overridden by pragma embedded in source code.

Option **-dm module**

This option is replaced by the `-bb` option. They have the same behavior now.

Option **-sm module_name**

This option indicates nLint to filter out all the violations detected in module, which is specified by `<module_name>`.

The difference between this option and `-dm` is that `-dm` will cause nLint not go into the module specified by `<module_name>`. But by using this option, nLint will still go into the module specified by `<module_name>` and do checking. If any violation is detected, nLint will filter out the violation under the specified module.

If the specified module contains instance, by using `-dm`, the module of the instance will not be checked. But by using `-sm`, the module of the instance will be checked.

Wildcard characters are also supported when specifying a module name.

The `-smr` option is used to specify the modules to be suppressed recursively.

Option **-bb module_name**

This option indicates nLint to treat the module specified as macro cell. All the submodules under the specified module will not be traversed into.

This option aims to ask nLint to ignore certain source codes, such as IP code, behavior code or code not written in HDL. All the information under the specified module will be gone except that the information of port connection will be kept for the module instance.

It is said that, nLint will connect the port instance to the instance of the specified module according to the module definition. But nLint will not traverse into the specified module.

Wildcard characters are supported when specifying a module name. For example, `ram*` will be used to represent `ram16x64`, `ram8x256`, etc; and `*ram` will be used to represent `my_ram`, `your_ram`, etc.

Option **-df file/-uf file_list**

This option indicates nLint to suppress the violations detected in the specified *file*. By using `-uf`, you can collect the design files want to suppressed into a file. nLint will first detect the violation and then suppressed it if the violation happens in the file to be suppressed.

Wildcard characters are supported when specifying a file name.

Detail message **-detail**

Some rules in nLint relate to path or some complex context. This option is used to tell nLint to print out the detailed information after a violation. Following is an example.

```
CCU.v(113): Warning 22131: clock signal should not be driven by
sequential logic (clock: "C5" (PCU.v(62)); sequential logic
output: "C5") (DFT,Design Style)
  Drive : system.i_cpu_1.i_CCU.C5
  system.i_cpu_1.i_CCU.C5 -> system.i_cpu_1.i_PCU.C5
    C5 -> ^
    i_CCU -> C5
    C5 -> i_PCU
    -> C5
```

The indented lines are detailed information for rule 22131.

Option **-opt filename**

This option is used to load an option file. In the option file specified by `filename`, all the command line options can be put in it, each for one line. Thus you can specify many options for nLint in a file and need not repeat typing for each run, such as the value setting, enable or disable rules, disable files, etc.

Here is an example: `-opt opt_file`, in `opt_file`,

```
-r 21001
+r 21003
-vs top.a=1'b1
-vs top.b=1'b0
-df ram1.v
-df ram2.v
```

Option **-vf format_string**

This option is used to specify the format of output message, which is also defined in the **Lint -> Message** tab of the **Tools -> Preferences** command in the nLint main frame window. For example: `"%f(%l): %t %n: %m %s(%g)";` it means that nLint will print out a message as follows:

```
"file(line):severity rule_number: module message groups"
```

Option **-ignore_pragma**

nLint provides pragma embedded in source code to enable/disable a rule. This option is used to suppress the effect of the pragma in source code.

Option for Partition

This set of options are used to set partitions of the design. The partition is a scope of the design. The following rules will be impacted by the partition setting:

- 25015, outputs leaving partition without been driven by register.
- 25011, input with heavy transitive fan out of end points.
- 25013, output with heavy transitive fan in of start points.

The input/output mentioned in the above rules are those input/output of the module as partitions.

Option `-partition_top`

This option is used to specify that only the top module is the partition of the design. This is the default setting.

Option `-partition_module`

This option is used to specify that all modules are the partitions of the design.

It will override `-partition_top` regardless of the order specified in the command line.

Option `-partition_scope`

This option is used to specify that all instances under the scope specified are the partitions of the design.

It will override both `-partition_top` and `-partition_module` regardless of the order specified in the command line.

Multiple `-partition_scope` options in the command line is allowable. All settings will be accumulated.

Option to specify clock source

There are some options related to specifying clock source for nLint

- `-vclk_source <vclk source>`: specify virtual clock source name.

- `-clk_source <signal>+VS_<name>`: specify clock source signal or assign clock source to virtual clock source.
- `-gen_clk_source NM_name+m/d<n>+p<n>+MA_master+PS_<pin/signal>`: specify generated clock source.

`-vclk_source` is used to specify a virtual clock source. The source name given does not exist in the design.

`-clk_source` is used to specify a real clock source. The full hierarchy name of a signal, port is acceptable. If with `+VS_<name>` option, it is used to specify the real clock source to belonging to a virtual clock source. This is useful to construct some real clock source under a same virtual clock source. Here the `<name>` is the specified virtual clock source.

`-gen_clk_source` is used to specify a generated clock source. In clock source extraction, there may be many reasons to cause the tool not to extract expected clock source tree. For example, there is a clock generated circuit is a block box. In this case, this option is useful to help the tool to construct the expected clock source. The detail usage of the option is as following.

`NM_<name>` is the generated clock source name.

`m/d<n>` is the frequency relationship between the generated clock source and the master clock source. 'm' means multiply. 'd' means divide. `<n>` is the number to multiply or divide. So the frequency of generated clock source equals to the frequency of master clock source to multiply or divide by `<n>`.

`p<n>` is the phase shift relationship between the generated clock source and the master clock source. `<n>` is the number of the phase of the master clock source to be shifted. After shifted, it is the phase of the generated clock source.

`PS_<pin/signal>` is the generated clock source signal or pin name.

Option for Beautify Output File

The `-beauty` option is used to beautify the output file of violations. When it is specified, the output messages will be organized with tree style, and the `-vrf` option will be suppressed.

A message line will be broken and indented in the next line when it is too long.

When combined with the `-sort` option, you can get different sorting results with beautified result.

Supposed that `-beauty` is specified and the result is sorted by `file`, you will get the following:

```
Total 56 Warning(s), 8 Error(s)
../src/pram.v -6 Warning(s)
  ../src/pram.v(33) :Warning : Rule 23002 : memory inferred
    on signal
    "macroram" not in library
  ../src/pram.v(37) :Warning : Rule 27335 : statement/
    expression used in translate_off/translate_on may cause
    mismatch between pre-synthesis and post-synthesis
...
ALUB.v -1 Error(s)
  ALUB.v(34) :Error : Rule 11000 : syntax error ->
    "module" CCU.v -6 Error(s), 29 Warning(s)
  CCU.v(52) :Error : Rule 25005 : signal "C21_C20" has never
    been assigned
  CCU.v(52) :Error : Rule 25005 : signal "C21_C20" has never
    been referenced
...
```

Option for Line Compiler Directive

In Verilog 2001, the line compiler directive is supported as follows:

```
line_compiler_directive ::=
  `line number "filename" level
```

We support this compiler directive except level.

If the `-lineMapping` option turns on, the file, line of the violations exported will be mapped to original source file, line according to the line compiler directive.

Option `-filter filter_file`

This option is used to specify a filter file with information in order to tell nLint to filter out some violations. The format of the filter file is as follows:

```
@nLint rc file Version 1.0
[Violation3]
Number = 22056
[Violation2]
File = ALUB.v
[Violation1]
Number = 22011,22013
File = PCU.v
```

It can be exported in graphical user interface of nLint. It can also be written by user. For the detailed format, please refer to the format of filter file.

This option will take effect in both GUI and batch modes. In GUI mode, the violations in report viewer will be filtered. In batch mode, the violations filtered out will not be exported to the ASCII file by `-out output_file`. In report DB, the violations filtered out are still there with just tagged as filtered out.

In batch mode, without specifying the `-filter` option, only those violations that are not filtered out will be exported from report DB. If the `-filter` option is specified, the filter out tag in the report DB will be ignored and apply the filter scheme in `filter_file` by using `-filter filter_file`.

Option for Return Value

If nLint is building in regression flow, the regression environment or script may require nLint to return a specified value when some conditions are encountered. The following are some settings that will change the return value of nLint.

Option `-on_rule_exit All|<rule_number>|AnyCompilerError`

nLint quits linting if violation is detected. “All” represents any rules; “rule_number” represents rule number; “AnyCompilerError” represents compiler error number. The return value is 1 by default.

Option `-ee exit_value`

nLint returns the specified value if any violation is detected.

With option `-on_rule_exit`, nLint stops linting and returns to a specified value if violation of rules is encountered.

Option `-active_group top_group`

This option is used to tell nLint to check the top group in RS file.

Option to create clock domain and source tree

The `-ex_clk` option is used to create a clock domain.

If `-ex_clk` is turned on, the clock domain information will be saved into *nLint* database file. Please refer to `-rdb` for more information. If `-ex_clk` is turned off, *nLint* will not extract clock domain and the clock domain related rules will not be checked.

Generally, the clock database can be generated under the graphical user interface (GUI) of Verdi or nLint. The database will be saved after clock domain extraction. However, if the design is changed, there may have mismatch between the generated clock database and design. In such case, nLint will discard the whole clock database file.

Option `-vclk_source <name>`

This option is used to specify the virtual clock source. You can specify some true clock sources which belong to the specified virtual clock source to construct the un-related clock source roots under the same virtual clock source root.

Option `-clk_source`

This option is used to specify the clock source. The syntax of this option is:

```
-clk_source <signal>+d|m<n>+p<n>+VS_<name>
```

In the arguments list, the `<signal>` is the port or signal in design, which is specified as a true clock source. The full hierarchy name is required.

If the specified clock source will belong to a virtual clock source, the other arguments in the list can be used.

The `d|m<n>` is used to specify the frequency of the clock source `<signal>` is divided (`d`) or multiplied (`m`) by `<n>` times of the virtual clock source. If it is not specified, the default value is `m1`, which means same frequency.

The `p<n>` is used to specify the phase shift relative to the virtual clock source. If it is not specified, the default value is `p0`, which means no phase shift.

The `VS_<name>` is used to specify the virtual clock source name. If it is not specified, it means the true clock source does not belong to any virtual clock source.

By specifying a clock source, you can explicitly construct a clock source tree with the specified clock source as root, except that the clock source specified belonging to a virtual clock source.

Option **-clk_uncern signal**

This option is used to specify the unconcern signal when resolving clock source. When tracing back to extract the clock source from a clock signal. Some simple gates will be encountered, i.e., AND. If nothing is specified, the resolving action will be stopped since there is more than one path to continue to trace backward. By specifying one of the signals of the input of the AND gate, nLint will continue to trace backward with the other input. It means that the signal specified can be ignored.

Any signal can be specified as an unconcerned signal if it is meaningful. nLint will continue to trace backward only if there is a possible path left.

When specifying the signal, a full hierarchy name is required.

Option **-gated_clk_cell <cell.port>**

This option is used to specify whether to pass through the specified `cell.port` if it is encountered when extracting clock domain. If the `<cell>` is encountered when extracting clock domain, nLint will pass through from the input `<port>` specified backward.

Option **-clk_lasr on/off**

This option is used to specify whether to treat the latch as register when extracting the clock domain. If it is turned on, the latch enable signal will be treated as clock. By default, it is ON.

Option **-clk_masr on/off**

Similar to the above option, this option is used to specify whether to treat the memory as register when extracting the clock domain. By default, it is ON.

Option to extract crossing path

The following options are used to specify to extract crossing path between interested clock domains.

Option `-clk_export filename`

This option is used to tell nLint to export clock domain information in the filename with ASCII format. Here is the example:

```
Source system.clock, Unresolved Comb Source
system.clock_posedge
  Clock Signal: system.i_pram.clock
  1 register(s) driven by this clock signal
system.clock_negedge
  Clock Signal: system.i_cpu.i_ALUB.clock
  2 register(s) driven by this clock signal
  Clock Signal: system.i_cpu.i_CCU.clock
  13 register(s) driven by this clock signal
Source system.i_cpu.i_ALUB.T3, Level0, phase 0
system.i_cpu.i_ALUB.T3_posedge
  Clock Signal: system.i_cpu.i_ALUB.T3
  1 register(s) driven by this clock signal
Source system.i_cpu.i_ALUB.T4, Level0, phase 0
system.i_cpu.i_ALUB.T4_posedge
  Clock Signal: system.i_cpu.i_ALUB.T4
  1 register(s) driven by this clock signal
Source system.i_cpu.i_CCU.C5, LevelN, phase unknown
system.i_cpu.i_CCU.C5_negedge
  Clock Signal: system.i_cpu.i_PCU.C5
  1 register(s) driven by this clock signal
Source system.i_cpu.i_ALUB.S1, LevelN, phase unknown
system.i_cpu.i_ALUB.S1_negedge
  Clock Signal: system.i_cpu.i_PCU.S1
  1 register(s) driven by this clock signal
Source system.i_cpu.i_CCU.C19, LevelN, phase unknown
system.i_cpu.i_CCU.C19_negedge
  Clock Signal: system.i_cpu.i_CCU.C19
  2 register(s) driven by this clock signal
Source system.i_cpu.i_ALUB.T2, Level0, phase 0
system.i_cpu.i_ALUB.T2_posedge
  Clock Signal: system.i_cpu.i_ALUB.T2
  2 register(s) driven by this clock signal
Source system.i_cpu.i_CCU.C6, LevelN, phase unknown
system.i_cpu.i_CCU.C6_negedge
  Clock Signal: system.i_cpu.i_PCU.C6
  1 register(s) driven by this clock signal
Source system.i_cpu.i_CCU.C1, LevelN, phase unknown
system.i_cpu.i_CCU.C1_posedge
  Clock Signal: system.i_cpu.i_PCU.C1
  1 latch(s) driven by this clock signal
```

Exceptionally, if the filename is "screen", nLint will print the above clock domain on screen. You can also specify the same file name as option -out. nLint will export both clock domain and violations into the same file.

Option -clk_export_reg

This option is used to tell nLint to export register information if -clk_export is specified. Here is the example. The registers connected to the clock signal are printed out.

```
Source system.clock, Unresolved Comb Source
  system.clock_posedge
    Clock Signal: system.i_pram.clock
    1 register(s) driven by this clock signal
    dataout
  system.clock_negedge
    Clock Signal: system.i_cpu.i_ALUB.clock
    2 register(s) driven by this clock signal
    IXR_tmp
    ACC_tmp
    Clock Signal: system.i_cpu.i_CCU.clock
    13 register(s) driven by this clock signal
    next_MA
    C21
    C20
    C19
    carry_mode
    bus_mode
    alu_mode
    CH
    C6
    C5
    mux_sel
    C1
    C0
Source system.i_cpu.i_ALUB.T3, Level0, phase 0
  system.i_cpu.i_ALUB.T3_posedge
    Clock Signal: system.i_cpu.i_ALUB.T3
    1 register(s) driven by this clock signal
    IXR
Source system.i_cpu.i_ALUB.T4, Level0, phase 0
  system.i_cpu.i_ALUB.T4_posedge
    Clock Signal: system.i_cpu.i_ALUB.T4
    1 register(s) driven by this clock signal
    ACC
Source system.i_cpu.i_CCU.C5, LevelN, phase unknown
  system.i_cpu.i_CCU.C5_negedge
    Clock Signal: system.i_cpu.i_PCU.C5
    1 register(s) driven by this clock signal
    IDR
Source system.i_cpu.i_ALUB.S1, LevelN, phase unknown
  system.i_cpu.i_ALUB.S1_negedge
    Clock Signal: system.i_cpu.i_PCU.S1
    1 register(s) driven by this clock signal
    PC
Source system.i_cpu.i_CCU.C19, LevelN, phase unknown
```

```

system.i_cpu.i_CCU.C19_negedge
  Clock Signal: system.i_cpu.i_CCU.C19
    2 register(s) driven by this clock signal
    IR7_IR2
    IR
Source system.i_cpu.i_ALUB.T2, Level0, phase 0
system.i_cpu.i_ALUB.T2_posedge
  Clock Signal: system.i_cpu.i_ALUB.T2
    2 register(s) driven by this clock signal
    carry_flag
    zero_flag
Source system.i_cpu.i_CCU.C6, LevelN, phase unknown
system.i_cpu.i_CCU.C6_negedge
  Clock Signal: system.i_cpu.i_PCU.C6
    1 register(s) driven by this clock signal
    TR
Source system.i_cpu.i_CCU.C1, LevelN, phase unknown
system.i_cpu.i_CCU.C1_posedge
  Clock Signal: system.i_cpu.i_PCU.C1
    1 latch(s) driven by this clock signal
    TDB

```


nLint Return Value

Overview

nLint uses different return values to indicate linting states. You can determine whether the linting process is normal or not by checking the nLint return value.

nLint Return Values

The return values are summarized as follows:

Return Values	Indication
0	Linting was successful, or nLint help is displayed (nLint -h or nLint -help).
1	"-on_rule_exit <rule>" is specified and the rule violation is found, but "-ee exit_value" is not used.
2	Failed to check out license.
3	Fail to import the design, or fail to initialize nLint.
4	The number of compilation errors exceeded the maximum compilation error number.
111	TCL command can't be interpreted or it executed with errors.

Please note that you can use the command line option "-ee exit_value" indicate a specific nLint return value if the specified rules are violated with the option "-on_rule_exit <rule>".

Rule Groups

Overview

nLint checks for more than 400 rules. The rules cover two major issues: design issues and readability issues. For details about each rule, refer to *nLint Rule Category*.

Rule groups related to design issues and readability issues are explained below.

Design Issues

Rules related to design issues can be divided into two rule groups: EDA tool usage and design and coding practices.

NOTE: Rules dealing with the design issues may appear in more than one group. For example, the rule to check for combinational feedback loops is under the Design Style rule group (it is not a good design style) and the DFT rule group (it cannot be handled by most ATPG tools.)

EDA Tool Usage

The following are rule groups for EDA tool usage:

- **Simulation:** Rules in this group check for the conditions that can affect the simulation negatively, both in design correctness and efficiency. It flags the part of code that is unconventional, or ridden with potential bugs. This rule group can be sub-categorized with respect to the nature of conditions it checks against.
- **Synthesis:** Rules in this group check for the conditions that are not appropriate during the synthesis process. Note that rules in this group can be bypassed by using the `// synopsys translate_on` and `// synopsys translate_off` directives in the code.
- **DFT:** Rules in this group check for the conditions that may affect the design testability, especially the efficient use of ATPG tools.
- **ERC:** Rules in this group check whether any electrical errors (e.g. floating nets) exist in the design.

- **VITAL Compliant:** Rules in this group check for compliance to the VITAL package.
- **Clock:** Rules in this group check for the clock domain structure that contains potentially anomalous.
- **CTS:** Rules in this group check for the clock tree which contain improper settings for clock tree synthesis.

Design and Coding Practices

The following are rule groups for design and coding practices:

- **Design Style:** Rules in this group check for inefficient, unconventional, or undesired design styles.
- **Language Construct:** Rules in this group check for strange, unconventional, undesired, or potentially anomalous language constructs in the source code.
- **HDL Translation:** Rules in this group check for the conditions that may affect the use of HDL translation tools (e.g. Verilog to VHDL or VHDL to Verilog translation tools).
- **Block Interconnect:** Rules in this group check for connection conflicts between blocks (e.g. the input-to-input connections).

Readability Issues

The following are rule groups about readability:

- **Coding Style:** Most rules in this group are related to readability. Usually violating the coding style rules will not cause serious errors in the design, but avoiding these violations makes the design more readable.
- **Naming Convention:** Many rules in this group contain the arguments that can be customized. For example, rule 21007 "signal name too long" checks for the length of signal names. The maximum length for these signal names can be customized. Use the **Rule Organizer** to customize these arguments.

NOTE: For the design issues, a rule violation can be potentially a serious error in the design or coding practices. Unlike the design issues, a violation in readability is less harmful. Hence, the checking for the two rule groups for readability issues is turned off by default. Use the **Rule Organizer** to turn on these two groups before they are checked by nLint.

Rule Setting File

Overview

nLint provides rule setting file (RS file) for user to configure the checking rules and arguments for each rule. With its multiple RS file loading support, you may load more than one RS file at once. Notice that all loaded RS files are merged before linting the design. We also support multiple top rule groups in one RS file.

RS File Format

The format of RS file is simple and readable. nLint provides a default RS file, `nLint.rs`, under the install directory. The RS file is divided into several sections, which will be described in the following paragraphs. Notice that section name is embraced with brackets [], such as `[section_name]`; and the items in the section are defined with the `key = value` style.

In the following sections, words in *Courier* type represent the words cited from RS file.

RS File Version

File Version Section

```
[RuleVersion]  
Version = 5004.06
```

This section describes the version of RS file (not the release version of nLint). For nLint 5.4v6, its rule version is 5004.06. This information is kept by nLint for backward supporting RS versions. It should not be modified by user.

Top Groups

Groups Section

```
[Groups]
1="nLint"
2=RMM
3="IPQ Design Guidelines"
```

NOTE: The group name should be embraced by quotes ("") if there are spaces within the name. If there is no quote, only the first word after equal sign (=) will be treated as the group name, which is not expected.

For example:

```
4 = Design style
```

The group name will be Design, but not "Design Style".

After that, all settings for a certain top group will be described in section with name [`<top group>.xxxx`].

For example:

```
[RMM.21001]
```

This section describes the properties of rule 21001 in top group RMM.

Default Checked Group Section

```
[Default]
Default = "nLint"
```

This section indicates that nLint checks the rules in the group specified with the value of `Default = .` In the example above, nLint will check the rules in "nLint". In each RS file, there is only one default checked group. If this section is not specified, the default checked group is the first top group specified under section [Groups].

Group Settings

Group Include Section

```
[nLint.include]
1 = "Simulation" Enable TRUE
2 = "Synthesis" Enable FALSE
3 = "DFT" Enable
4 = "Design Style" Enable
5 = "Language Construct" Enable
6 = "HDL Translation" Enable
```

```
7 = "Coding Style" Disable
8 = "Naming Convention" Disable
9 = "VITAL Compliant" Disable
```

This section indicates the including groups in default checked group with the group name in pattern "<top_group>.include". In the example above, the top group "nLint" includes the following sub-groups: "Simulation", "Synthesis", etc.

On the other hand, the second value `Enable` or `Disable` also indicates the status of the including groups. Assumed that the current default top group is "nLint". If `Enable`, nLint will check all rules included in the group. If `Disable`, nLint will not check the rules included in the group.

The third value `TRUE` or `FALSE` can also indicates the configuration of `Enable/Disable` status for the including groups. If `TRUE`, the `Enable/Disable` status will be configurable. If `FALSE`, the status will depend upon the second value. However, the third value is not essential since its default value is `TRUE`.

Notice that *configurable* means that when applying multiple RS files merge scheme, the setting can be overwritten by the latest RS file; while *non-configurable* means that the setting cannot be overwritten by latest RS file.

For different top groups, an included group can be named the same as following:

```
[nLint.include]
1 = "Simulation" Enable TRUE
...
[Gate.include]
1 = "Simulation" Enable TRUE
```

But the "Simulation" group in "RTL" and "Gate" are different. They are two sub-groups.

Rules in Sub-group

```
[Simulation]
22005 = Enable Enable TRUE TRUE
22011 = Enable Enable TRUE FALSE
22013 = Enable Enable FALSE FALSE
...
```

This section indicates the rules belong to a particular group with its group name. Notice that the rule number is used as a key.

The first value `Enable`, `Disable` or `NONE` means that the rule is enabled, disabled or not supported in the group for Verilog. If `Enable`, nLint will check the rule if the group is also enabled in the checked group on Verilog HDL source code. If `Disable`, nLint will not check the rule even if the group is enabled in the checked group. If `NONE`, nLint does not support the rule for Verilog.

The second value indicates the availability of rules for VHDL source code.

The third value indicates the configuration settings of rule properties for Verilog. If `TRUE`, the rule can be enabled or disabled in the merged file. If `FALSE`, the rule will keep its default value (always enabled or disabled). The fourth value is similar to the third value except that it is used for VHDL. However, the third and fourth values can be omitted since the default value is `TRUE`.

NOTE: The third and fourth values are not related to the configuration settings of rule argument, which will be described later in the following sections.

Parameter for Special Rule

```
[nLint.21013]
vhdl_val="PREFIX", "clk_"
vlog_val="PREFIX", "clk_"
vlog_severity = Level2
vlog_changeable = TRUE
vlog_TranslateOff = OFF
vhdl_severity = Level2
vhdl_changeable = TRUE
vhdl_TranslateOff = OFF
index_name = "4-3-2-1"
vlog_description = "clock signal prefix"
vhdl_description = "clock signal prefix"
```

The section name composed by the top group name and the rule number, `<top group>.<rule number>`.

This section indicates the rule parameter with rule number. The `vhdl_` or `vlog_` prefix in the keys used to indicate the values after equal sign (=) is applied to VHDL or Verilog. In nLint, rules are allowed with different parameter values for different language.

The parameter value should be specified by the Rule Category description for each rule.

The description for each key is illustrated as follows:

- `val`: Rule value, maximum two values; displayed as **Argument1** and **Argument2** in GUI. If there is more than one word in the value string for one argument, the value string must be quoted.
- `severity`: Rule severity, the default value is `Level2`; displayed as **Severity** in GUI.
- `TranslateOff`: Specify if the rule checking can be affected by translate directives. The value is either `ON` or `OFF`. If `ON`, the rule can be ignored by

translate directives. If `OFF`, the rule will not be affected by translate directives; displayed as **TO** in GUI.

- `changeable`: Indicates the configuration settings of rule argument. If `TRUE`, the rule argument will be configurable. If `FALSE`, the rule argument is not configurable.
- `index_name`: Specify the index alias, which is a mapping value, to replace the rule number shown in Rule Organizer. For example, the RTL and RMM groups may share a same rule; but for the index, the rule may be different in different groups. If it is not specified, the `index_name` is the string of rule number. If it is specified, the string will be shown in the place of rule number in GUI and warning message. Different rule number can map to a same `index_name` string.
- `description`: Specify the short description shown in GUI. For example, the RTL and RMM group may share a same rule; but for description, the rule can be different in different groups. If it is not specified, the short description is the string provided by the program. If it is specified, the string will be shown in the place of short description in GUI and warning message.

Compiler Errors

In this version, the error and warning messages reported by compiler will be included in nLint rules set.

Considering that compilation/elaboration error is a special group, mostly

- all top groups will include it
- the rule is simple, only allow user to enable/disable and change severity

So the compilation/elaboration errors will be grouped into a special top group - *Compilation/Elabration*. In this group, no sub-group is included and there are only those rules from compiler and elaborator. It cannot be included in other top groups. In the application, it will always be checked automatically. It is not necessary for user to choose this group to be checked. The rule number of this group is from 10000 to 20000.

For the rule format in this group, there is one rule for each language type and it is without any argument. Only the following properties can be changed.

```
index_name = "4-3-2-1"
```

If Verilog

```
vlog_severity = Level2  
vlog_changeable = TRUE  
vlog_description = "clock signal prefix"
```

If VHDL

```
vhdl_severity = Level2  
vhdl_changeable = TRUE  
vhdl_description = "clock signal prefix"
```

Only the rule with properties different to the default value, the rule property will be saved in a RS file.

Since the rule can only be included in the special group, the enable/disable setting is also recorded as a property of the rule. That is, in RS file, the rule setting will be as follows:

```
[Compilation/Elaboration.16021] #failed to find identifier %s"  
Error, by default  
vlog_severity = Level2  
vlog_description = "unknown identifier"  
enable = FALSE
```

Merge Criteria

If there is more than one RS file selected, nLint will merge them together before linting. In general, the setting in last RS file will overwrite the same setting of the previous RS file. However, nLint also provide an argument (configurable or non-configurable settings) in the RS file in order to specify the item dominated by the previous RS file.

Top Groups Section

If the top group specified in the second RS file is not in the first RS file, the merged RS file will include it.

The effect of Merging Top Groups Section is illustrated in the following table.

RS1	RS2	Merged result
[Groups]1 = RTL1	[Groups]1 = RTL1	[Groups]1 = RTL1
[Groups]1 = RTL1	[Groups]1 = RTL2	[Groups]1 = RTL12 = RTL2

Default Group Section

The default group section in merged RS file will follow the last RS file.

Group Include Section

If a sub-group under a certain top group in second RS file is not in the top group of the first RS file, the merged top group will include it.

The effect of Merging Top Groups Section is illustrated in the following table.

RS1	RS2	Merged result
[nLint.include]1 = "Simulation"	[nLint.include]1 = "Simulation"	[nLint.include]1 = "Simulation"
[nLint.include]1 = "Simulation"	[nLint.include]1 = "Design Style"	[nLint.include]1 = "Simulation" 2 = "Design Style"

If the configurable value of the previous RS file is FALSE, then the group, either is enabled or disabled, will be dominated by this file. Otherwise, the setting is

dominated by last RS files. The effect of merging Group Section is illustrated in the following table.

RS1	RS2	Merged result
[nLint.include]1 = "Simulation" Enable TRUE	[nLint.include]1 = "Simulation" Enable TRUE	[nLint.include]1 = "Simulation" Enable TRUE
[nLint.include]1 = "Simulation" Enable TRUE	[nLint.include]1 = "Simulation" Disable TRUE	[nLint.include]1 = "Simulation" Disable TRUE
[nLint.include]1 = "Simulation" Enable FALSE	[nLint.include]1 = "Simulation" Enable TRUE	[nLint.include]1 = "Simulation" Enable FALSE
[nLint.include]1 = "Simulation" Disable FALSE	[nLint.include]1 = "Simulation" Enable TRUE	[nLint.include]1 = "Simulation" Disable FALSE

Rules In Group

If a rule under a certain sub-group in second RS file is not in the sub-group of the first RS file, the merged sub-group will include it. The effect of Merging Rules In Group Section is illustrated in the following table.

RS1	RS2	Merged result
[nLint.simulation]22005 = Enable Enable	[nLint.simulation]22005 = Enable Enable	[nLint.simulation]22005 = Enable Enable
[nLint.simulation]22005 = Enable Enable	[nLint.simulation]22007 = Enable Enable	[nLint.simulation]22005 = Enable Enable 22007 = Enable Enable

If the configurable value of one rule is FALSE in the previous file, then the rule, either is enable or disable, in the group will be dominated by this file. Otherwise, the setting is dominated by last RS files. The effect of merging rules in Group Section is illustrated in the following table.

RS1	RS2	Merged result
[nLint.simulation]22005 = Enable Enable TRUE TRUE	[nLint.simulation]22005 = Enable Enable TRUE TRUE	[nLint.simulation]22005 = Enable Enable TRUE TRUE
[nLint.simulation]22005 = Enable Enable TRUE TRUE	[nLint.simulation]22005 = Disable Enable TRUE TRUE	[nLint.simulation]22005 = Disable Enable TRUE TRUE
[nLint.simulation]22005 = Enable Enable FALSE TRUE	[nLint.simulation]22005 = Enable Enable TRUE TRUE	[nLint.simulation]22005 = Enable Enable FALSE TRUE
[nLint.simulation]22005 = Disable Enable FALSE TRUE	[nLint.simulation]22005 = Enable Enable TRUE TRUE	[nLint.simulation]22005 = Disable Enable FALSE TRUE

Special Rule Parameter

If the configurable value (`vlog_changeable`) in a previous RS file is `FALSE`, all rule arguments for Verilog will be dominated by this file. Otherwise, the setting is dominated by the last RS files. For VHDL, the configurable value is decided by `vhdl_changeable`. The effect of merging rule parameters is illustrated in the following table.

RS1	RS2	Merged result
[nLint.21013]vlog_val="PRE FIX", "clk_"vlog_severity = Level2vlog_changeable = TRUEvlog_TranslateOff = OFF	[nLint.21013]vlog_val="PRE FIX", "clk_"vlog_severity = Level2vlog_changeable = TRUEvlog_TranslateOff = OFF	[nLint.21013]vlog_val="PRE FIX", "clk_"vlog_severity = Level2vlog_changeable = TRUEvlog_TranslateOff = OFF
[nLint.21013]vlog_val="PRE FIX", "clk_"vlog_severity = Warningvlog_changeable = FALSEvlog_TranslateOff = OFF	[nLint.21013]vlog_val="PRE FIX", "clk_"vlog_severity = Warningvlog_changeable = TRUEvlog_TranslateOff = OFF	[nLint.21013]vlog_val="PRE FIX", "clk_"vlog_severity = Level2vlog_changeable = FALSEvlog_TranslateOff = OFF
[nLint.21013]vlog_val="PRE FIX", "clk_"vlog_severity = Level2vlog_changeable = TRUEvlog_TranslateOff = OFF	[nLint.21013]vlog_val="PRE FIX", "c_"vlog_severity = Level3vlog_changeable = TRUEvlog_TranslateOff = ON	[nLint.21013]vlog_val="PRE FIX", "c_"vlog_severity = Level3vlog_changeable = TRUEvlog_TranslateOff = ON
[nLint.21013]vlog_val="PRE FIX", "clk_"vlog_severity = Level2vlog_changeable = FALSEvlog_TranslateOff = OFF	[nLint.21013]vlog_val="PRE FIX", "c_"vlog_severity = Level3vlog_changeable = TRUEvlog_TranslateOff = ON	[nLint.21013]vlog_val="PRE FIX", "clk_"vlog_severity = Level2vlog_changeable = FALSEvlog_TranslateOff = OFF

Suppress a Violation

Overview

This document describes all the aspects that will affect the report of a violation in nLint. This includes to disable a rule checking and to suppress to report a rule.

Set Up Lint Entry

In general the design imported may include the test bench. It is easy to specify only linting the design under analysis by option `-lintTop`. In this way, the test bench is ignored.

Disable/Enable a Rule

This section describes how to disable/enable a rule checking or a set of rule checking based on the nLint functional order.

Initial All Rules Enable/Disable Statically

Since there may have more than one RS file specified for nLint, nLint needs to merge all RS files first. You can specify the RS files and affect the merge action by the following options.

`-rs rs_files`: Specify rule setting files.

`-nors install:home:current`: Do not merge the specified rule setting files.

`-drm`: Do not merge the rule setting files; only the last one will take effect under this option. For a rule, it may be included in more than one group.

You can specify the active top group to be checked by following option.

`-active_group <top_group>`: Specify to check the active top group by `<top_group>`.

You can specify to check only the rules with severity you wanted by following options.

`-maxseverity severity_str`: Specify maximum severity of warning to be shown.

`-minseverity severity_str`: Specify minimum severity of warning to be shown.

After resolving all RS files, we get a result that for each rule, it is enabled or disabled globally.

In the command line, you can also enable or disable some rules globally, regardless of their status in RS file.

`-r rules`: Specify the rules not to be checked. It will override the pragma in the design source file. It will override the rule setting file.

`+r rules`: Specify the rules to be checked. It will override the pragma in the design source file. It will override the rules setting file.

`-pr rules`: Specify the rules not to be checked. It will not override the pragma in the design source file. It will override the rule setting file.

`+pr rules`: Specify the rules to be checked. It will not override the pragma in the design source file. It will override the rule setting file.

You can specify more than one rules a time, such as `-r 21001:21003` (on WIN32 platform, you should use `-r 21001;21003` instead).

The `<rules>` can be a group. For example, `-r Simulation` means to disable all rules under the Simulation group. It is similar to specify all rules under *Simulation* followed `-r`. There is a special group named *All*, which is used to indicate all rules in nLint.

The command option `+r/-r` will override the setting of RS file.

After that, we get a table for each rule that it is disabled or enabled globally.

Import Design

nLint can compile the design from source code or load the design from library. If compile the source code in, there may have compilation/elaboration violations. To suppress the violations depends the rule of the violations is disabled or enabled. Even though the rule is enabled, the following option can suppress it also.

`-no_syntax`: Ignore compiler errors.

And the violations are kept in a temporary storage and they will be reported after the step - [Resolving Rule Disable/Enable by Design Context](#).

Resolving Rule Disable/Enable by Design Context

The design context means the design files, modules of design hierarchy and the detail line of the source files. In resolving the rule status, the following options will be considered.

- df file_name: Specify ONE file not to be checked.
- uf file_name: Specify a file containing a list of file names not to be checked.
- sm module_name: Suppress the specified module.
- smr module_name: Suppress the specified module recursively.
- bb module_name: Tell the tool not to traverse into the specified module.
- ignore_pragma: Ignore Novas pragma in source code.

After resolving the status, nLint create a table for each source file with the information that, at a place of design, which is represented by file/line, a rule is disabled or enabled.

Initially, the rule status of disable/enable of each file comes from the first step - [Initial All Rules Enable/Disable Statically](#).

In resolving the pragma used in source code, when encountered a plus sign (+), nLint will enable the specified rules. When encountered a minus sign (-), nLint will disable the specified rules. After scanning a file, the pragma in this file will not take effect any longer.

In resolving -df/-uf, all rules at the beginning of the specified file will be disabled.

In resolving the -sm option, nLint will:

- keep the status at the beginning of the module for all rules;
- disable all rules at the beginning line of the module;
- restore the status kept in the beginning of the module.

For VHDL design, nLint will check if each source file is in the current working library or in the library the top module located. If not, nLint will disable the file like -df. The following option may change the setting:

`-lintlib logic_lib_names`: Specify the logic libraries to be linted. By default, only those libraries in the current working library will be linted.

It will cause nLint to check if the file is also located in the specified library, just like to check the file in current working library.

Traversing Design

Generally, the following are the steps for nLint to check all violations.

- Depth-first traverse the design
 - Check declaration
 - Check statement
- Check global rules

In this step, the following option will impact the behavior of nLint.

`-bb module_name`: Tell the tool not to traverse into the specified module.

After the design is imported, nLint will traverse the hierarchy of the design and get all modules under the specified `<module>` and keep them. In traversing step, nLint will not go into the modules it got. So there must not have any violation for modules resolved by the `-bb` option.

Checking

Generally, nLint can identify how an object appears in the design (in a file or line). So nLint can check the table comes from [Resolving Rule Disable/Enable by Design Context](#) to know if a rule needs to be checked on the object. If a rule is disabled on the object, nLint will not check it.

But in some complex rules like *combinational loop*, before checking, nLint does not know the violation will happen on which object. So nLint needs to do checking first. After nLint identifies a violation on a special place, nLint will check the table to see if the rule is disabled to report the violation.

In this step, some options will impact the scheme.

`-lint_cell_lib`: Turn ON to lint library files. OFF by default.

When encountering a lib cell for Verilog design, nLint will check if the switch is turned on. If not, nLint will disabled the rule checking.

`-wn warn_num`: Specify maximum number of warnings for this run.

`-pwn warn_num`: Specify maximum number of warnings per rule.

When reporting violations, nLint will remember how many violations reported on each rule. Before reporting another new violation, nLint will check if the warning number of `-pwn warn_num` is encountered. If the number is reached, nLint will disable the rule checking.

When nLint would go into a new module, nLint will check if the total violations number by `-wn warn_num` is encountered. If encountered, nLint will stop traversing and finish the checking.

UDR Checking

After nLint checked all built-in rules, it will enter UDR checking. When user-defined action reports a violation to nLint violation center, nLint will check whether the violation happens where with the table got from [Resolving rule disable/enable by design context](#) to decide to report the violation, just like the built-in rules.

The following option will impact the checking behavior.

`-udronly`: Indicate nLint only check user-defined rules.

It tells nLint to skip checking the built-in rules. It is said that by this option, nLint will skip the steps [Traversing Design](#) and [Checking](#).

`-noudr`: Indicate nLint not to check user-defined rules.

It tells nLint to skip UDR checking. So there will be no UDR violations.

Filter Out Violations

This section describes how the filter setting file works.

After nLint checking, all the violations will be saved into report DB. However, you can still apply filter scheme to filter out some violations when exporting the report DB into report files.

The following option is used:

`-filter <file>`: Specify the filter setting file.

Filter File Format

The following paragraphs describe the format of filter file for the user who wants to read/edit the file.

In this file, you can define your own filter setting. Each filter setting must have a name and nine fields (maximum), which are Description, Group, Number, File, Line, Object, Scope, Value and Related.

Name is used as an identifier of a setting. It must be unique in one filter file, and this field must be filled.

Description is used to describe what does this setting do, and it will not affect the comparison of violations.

Group means the rule group, its value may be Simulation, Synthesis, DFT, Design Style, Language Construct, HDL Translation, Coding Style, Naming Convention, VITAL Compliant or Clock.

Number means the rule number such as 22011, 21001, etc.

File is the file name of the violation.

Line is line number of the violation.

Object is the key object of the violation. Its values must be enclosed by double quotes (") if there are special characters (i.e. spaces) in it.

Scope is the scope that the violation belongs to.

Value is used to store some integer values for special violations. User is not recommended to edit this field. It is used for the *filter this violation* function in nLint.

Related is used to store some related information for filter. Its values may be enclosed by double quotes ("). User is not recommended to edit this field. It is used for the *filter this violation* function in nLint.

Notice that,

All fields can be empty except *Name*, but there must be at least one field has value, or the filter setting will be invalid.

Except *Description*, all fields can be set with multiple values, and each value may be separated by semicolon (;).

User is strongly recommended NOT to modify the value in *Value* and *Related* because the value in these two fields is not easy to understanding for user sometimes. And it will cause false filtering result if these fields are modified illegally.

However, user CAN decide to enable or disable these fields by setting a symbol T (TRUE) or F (FALSE) at the end of the fields. This is not supported in the current version.

We will support asterisk (*) in the *File*, *Object* and *Scope* fields. This means that you can write ROM* to match any object begins with ROM. Also, the regular expression will be supported in some fields if possible.

Make the *Value* and *Related* fields more readable for user so that user can modify them to get a more precise filtering result.

Some examples are provided for reference. These examples can touch most of the syntax phenomenon of the filter file. The function of the filter setting is described in the *Description* field of each filter item.

```
@nLint rc file Version 1.0
[DFT_PCU_error_reset]
Description="Filter DFT violations in sys/PCU.v related to
error_reset"
Group=DFT
File=sys/PCU.v
Object="error_reset"
[22001_22011_system]
Description="Filter violations of rule 22001 and 22011 in the
scope system"
Number=22001,22011
Scope=system
[22029_L13_ALUB]
Description="Filter violations of rule 22029 at line 13 of sys/
ALUB.v"
Number=22029
File=sys/ALUB.v
Line=13
[22003_data_out]
Description="Filter violations of rule 22023 related to data_out
which left object width is 2 and right object width is 4"
Number=22003
Object="data_out"
Value=2,4,T
[28001_clk_clock]
Description="Filter violations of rule 28001 which cross from
clock domain clk_RISING to clock domain clock_FALLING"
Number=28001
Related="clk_RISING","clock_FALLING",T
```

Applying Filter Setting

In nLint, the filter setting will not impact the violations that are saved to report DB.

When applying filter setting at checking, the violations being filtered will be tagged in report DB. In batch mode, you can export the violations not be filtered

into report file. In GUI mode, you can view the un-filtered violations, filtered violation or both in the report viewer by choosing commands in the report viewer.

If you want to generate a report by applying a new filter setting, use the following command line option:

```
nLint -rdb <report db> -filter <filter_file> -out <output_file>
```

Of course, if nothing is defined in <filter_file>, all violations will be exported into the <output_file>.

Clock Source Tree

Overview

This document explains the concept of clock source tree and its relevant features.

Terminology

Following are key terms in clock source tree and its relevant features.

Clock Signal

Clock signal is a signal directly connected to the clock pin of storage element in the design. The storage element can be flip-flop, latch or other memory storage element.

Internal Clock Source

An internal clock source is a signal, which drives clock signals that are defined in Clock Signal, by passing through their logics. The logic being passed through is optional. Usually, they are buffer, inverter, reduced buffer or inverter after constant value propagation. The internal clock source is the signal closest to the primary input port that drives the clock signals by passing through design hierarchy and the optional logic.

Clock Domain

A clock domain consists groups of storage elements. Two clock domains are under an internal clock source. One is triggered by positive edge of the internal clock source. The other is triggered by negative edge of the internal clock source. All the flip-flops, latches or other storage elements in a clock domain are triggered by the same edge of the internal clock source.

Clock Source Tree

Clock source tree is a tree structure. The node of it is an internal clock source. If an internal clock source is derived from another internal clock source by passing through certain logics, it is a child node of the internal clock source.

If an internal clock source is derived from its parent, there will be two derivation types.

If an internal clock source is derived from its parent by passing through combinational logic, derivation of an internal clock source to its parent is defined as "Level0", which means that it is a gated clock. See following figure for demonstration.

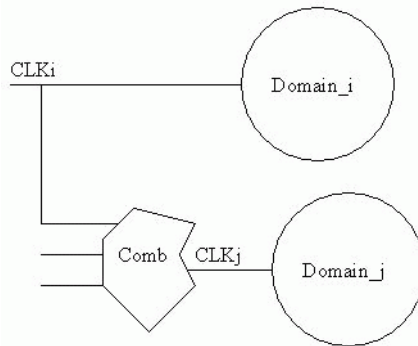


Figure: Parent clock source and child clock source with Level0

In the figure, there are two internal clock sources, CLK_i with clock domain $Domain_i$ and CLK_j with clock domain $Domain_j$, while $Comb$ represents combination logic except buffer and inverter. The internal clock source CLK_j is generated by CLK_i which passes through combination logic. So, internal clock source CLK_j is the child of internal clock source CLK_i in the clock source tree. Level0 is given to indicate that the internal clock source CLK_j is derived from its parent by passing through combinational logic.

Phase shift of an internal clock source to its parent is another property. For Level0 type child internal clock source, the phase difference to the parent internal clock source should be extracted. In the following figure, clock source CLK_1 has $T/2$ phase shift comparing to the parent internal clock source CLK .

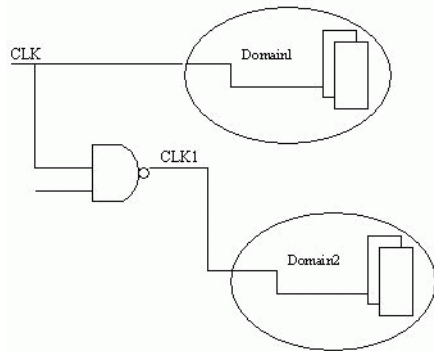


Figure: Phase shift between clock sources

If an internal clock source is generated from a combinational output of some flip-flops, which are triggered by the parent internal clock source, the derivation type of the internal clock source to its parent is defined as `LevelN`, which means that it is a generated clock. See following figure for demonstration.

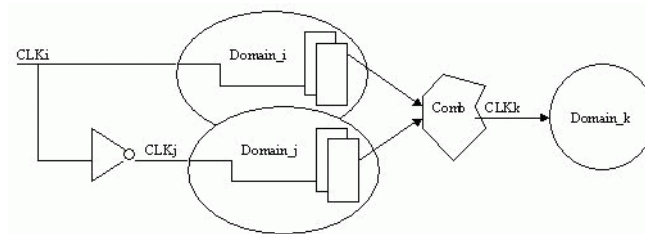


Figure: Parent clock source and child clock source with LevelN

In the above figure, there are two internal clock sources, `CLKi` with clock domain `Domain_i`, rising edge triggered, and `CLKj` with clock domain `Domain_j`, falling edge triggered. The internal clock source `CLKk` is generated by a combination logic of which the inputs are the output of flip-flops in `Domain_i` or `Domain_j`. The internal clock source `CLKi` does not pass through the combination logic to generate clock source `CLKk` directly. So the tool can't extract such relationship automatically. But the tool provides interface to specify the relationship.

Under the guide from user, in the clock source tree, clock source `CLKk` could be a child of clock source `CLKi`. `LevelN` is defined to indicate the derivation type of the internal clock source `CLKk` to its parent.

For `LevelN` type, the phase shift of an internal clock source to its parent cannot be extracted automatically. They will follow the specification from user.

Clock Source

Root of a clock source tree is a clock source. Resolved clock source can be a primary input, a specified clock source or an output of register. Floating wire and output of combinational logic are un-resolved clock source. If more than one clock source drives an internal clock source, an internal clock source is treated as multiply-resolved. A multiply-resolved internal clock source can be a root of a clock source tree.

Following is an example of clock3.v to demonstrate terms above. Assume that the test_s_clk is a specified clock source.

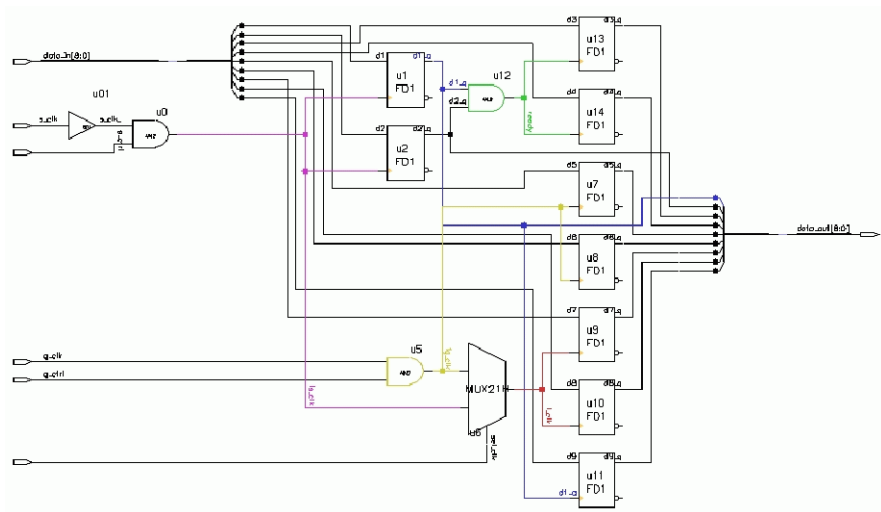


Figure: Example

The extracted clock source tree is as following.

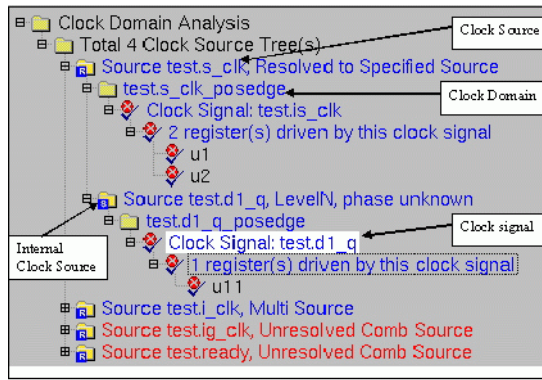


Figure: Example of Clock Source Tree

Let's focus on the expanded tree with the root `test.s_clk`.

The `test.s_clk` is specified as a clock source, so it is a root of the clock source tree.

Under the root, there is the internal clock source `test.is_clk`, which triggers two flip-flops `u1` and `u2` with positive-edge. The internal clock source `test.is_clk` is derived from `test.s_clk` by passing a buffer `u01` and an AND gate `u0`. So it is the child of `test.s_clk` with a derivation type `Level0` without phase shift. Under the internal clock source `test.is_clk`, a domain is triggered by `test.is_clk` with positive edge. The name of a clock domain is given based on an internal clock source and trigger edge of the internal clock source. The clock signal is also `test.is_clk`, under which two flip-flops, `u1` and `u2` are included.

Under `test.is_clk`, there is the internal clock source `test.d1_q`, which is the output of flip-flop `u1` triggered by `test.is_clk`. The derivation type therefore is `LevelN`.

Methods To Extract Clock Source Tree

Clock analyzer is used as an engine to extract a clock source tree. There are two steps in the engine to extract clock source tree:

- Extract clock domain with internal clock source.
- Build a clock source tree based on the clock domain and extracted internal clock source.

Constant Value Propagation

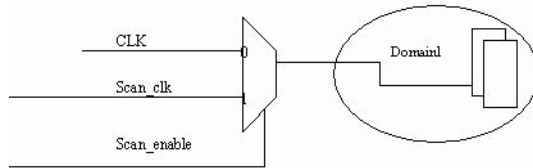


Figure: Example for constant value propagation

In the above figure, if we specify `scan_enable` as 0, the MUX will be reduced to a buffer (or inverter) with only one input `CLK`. If we specify `scan_enable` as 1, the MUX will be reduced to a buffer with only one input `scan_clk`.

Extract Clock Domain

The engine will collect all flip-flops (and latches) and find its clock signal. All flip-flops (and latches) of the same clock signal are merged into a group. Then the engine will trace backward to find the internal clock source from the clock signals.

After the engine has located the internal clock source, all the groups of flip-flops with the same internal clock source will be merged into a maximum of two clock domains under the internal clock source. One is a domain to be triggered by positive edge of the internal clock source that is located by the engine. The other is a domain to be triggered by negative edge of the internal clock source that is located by the engine.

When finding the internal clock source of the flip-flop (and latches) groups, there are some options to guide the engine to pass through some kind of logic.

Pass Through Logic

When the internal clock source of a group of flip-flops (and latches) is located, some logics can be passed through like a wire connection, which can be configured.

- Pass through assignment (by default ON)
- Pass through buffer (by default ON)
- Pass through reduced buffer (by default ON). The "reduced" here means that the logic is reduced with constant value setting.
- Pass through inverters (by default ON)

- Pass through specific gated clock cell (by default ON). The specific gated clock cell here means some special cell in library. In the cell, there is input clock pin and output clock pin. The cell is designed as a gated cell for gated clock to reduce the power consumption without causing any timing problem. Then, the engine continues to trace input clock pin when output clock pin is met.
- Pass through cell.port as an additional solution for gated clock cell. The input port of the cell.port is like the input clock pin in the gated clock cell mentioned earlier.
- Pass through cell with one input port and one output port (by default ON).
- Pass through cell with one input port after specifying non-concerned signal (by default ON).
- Pass through pure combinational gate (by default OFF)

When any of the above option is turned on, the engine continues to trace backward from the input of the logic that has been met.

When any of the option is turned off, the engine stops at the output of the logic and treats the output of the logic as the internal clock source.

Build Clock Source Tree

After extracting clock domain, the engine builds the clock source tree based on the result. The engine locates the clock source and develops the parent-child relationship for all internal clock sources under the same clock source.

For each internal clock source signal, the engine traces its fan in cone. In tracing, the engine stops at

- primary input
- floating wire or output port of unknown logic
- flip-flops output
- specified clock source

In the path to fan in cone, some internal clock sources are extracted in the first step. Afterwards, the engine constructs the parent-child relationship according to the derivation type and then extracts the phase shift between them.

After the root of the clock source tree is located, the clock source is resolved when the clock source is

- primary input,
- flip-flops output
- specified clock source.

The clock source is unresolved when it is floating wire or output port of complex (or unknown) logic.

The clock source is multiply-resolved when it is driven by more than one clock source.

Specify Clock Source

When building clock source tree, user can specify a clock source to guide the engine. It is not obligatory, but in some cases, it facilitates the engine to locate the correct clock source when a cell meets more than one input.

The primary port or any signal in the design can be specified as clock source with a full hierarchy name.

The specified clock source is a root of a clock source tree. This allows user to differentiate clock sources under the same root.

Specify Generated Clock

When building clock source tree, user can specify a signal or of pin of a cell as clock generated from a master clock source. In most cases, the clock generation circuit can not be recognized by the tool.

Virtual Clock Source

When an input design is not a complete design but part of the design, such as a block, there should be more consideration to complete the clock source tree.

If more than one clock source is input into a block of design, the clock sources may be derived from the same root. In this case, user might be interested in constructing the clock source under a same non-existence, "virtual" root. Following figure demonstrates this concept. Assumed that `clock1`, `clock2` and `clock3` are three clock sources of the block of design under analysis. Outside the block, they are derived from the same root `clock` by a clock divider.

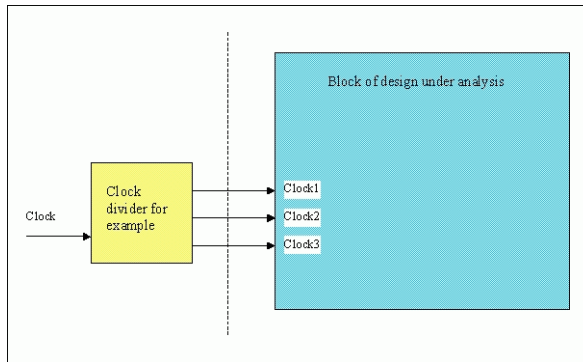


Figure: Specify virtual clock when clock divider is not of design

For this case, we can specify `clock` as a virtual clock source. And then specify `clock1`, `clock2` and `clock3` as children of the virtual clock source `clock`. For each child clock source, the frequency relationship and phase shift to the virtual clock source can be specified. The frequency relationship indicates the frequency of virtual clock source multiplying or dividing with an integer. The phase shift is indicated in a degree number. Generally, 0, 90, 180 and 270 are supported.

Some primary input ports of a block come from flip-flops of a clock source that does not exist in the under-analysis block. Or some primary output ports of a block may go to flip-flops of a clock source that does not exist in the under-analysis block. In these cases, user might be interested in constructing the primary ports under a non-existent, "virtual" clock source.

Following figure demonstrates this concept. Primary inputs (PIs) come from flip-flops triggered by clock source `clock_i` which is not in the block of the design under analysis. The `POs` goes to flip-flops triggered by clock source `clock_o` which is not in the block of design under analysis.

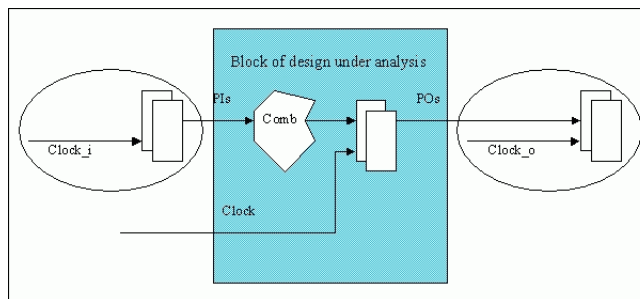


Figure: Specify primary ports associated with virtual clock

For this case, specify `clock_i` as virtual clock source with primary inputs `Pis` to represent the flip-flops triggered by `clock_i` outside the block of design under analysis. `clock_o` can be specified as another virtual clock source with primary outputs `Pos` to represent the flip-flops triggered by `clock_o` outside the block of design under analysis.

Specify Design Under Analysis

The design with test-bench (written in HDL) usually is represented as the following figure in HDL files (design hierarchy).

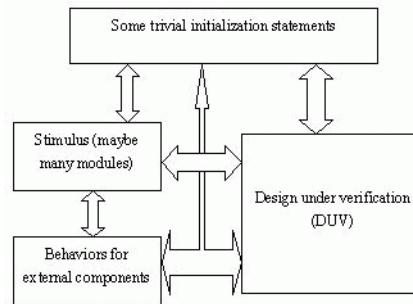


Figure: The layout of modules in the top of the design, one DUV

For analysis, a critical concern is the designing, not the code of stimuli nor external components, which may be non-synthesizable and cannot be analyzed correctly. And for design under analysis, the boundary like primary port is not in the top module as shown in the figure.

In some cases, user may not cut the code of design under analysis to analyze it, so user can specify the design under analysis. When design under analysis is specified, the engine only analyzes code in the design part and ignores code not in design part.