DesignChecker User Guide

Software Version 2010.3

June, 2011

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS CORPORATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

RESTRICTED RIGHTS LEGEND 03/97

U.S. Government Restricted Rights. The SOFTWARE and documentation have been developed entirely at private expense and are commercial computer software provided with restricted rights. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement provided with the software pursuant to DFARS 227.7202-3(a) or as set forth in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable.

Contractor/manufacturer is:
Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: www.mentor.com/supportnet
Send Feedback on Documentation: www.mentor.com/supportnet/documentation/reply_form.cfm

Table of Contents

List of Tables

Chapter 1
About DesignChecker

This chapter provides an overview of the key features of the DesignChecker static checking tool and its user interface.

Overview

This user guide describes DesignChecker checking and analysis tool.

An HTML version of this user guide can be accessed from the **Help** menu in the main DesignChecker window.

You can click on the  icon at the top of each HTML page in the user guide to access a printable Adobe Acrobat portable document format (PDF) version.

Other DesignChecker documentation which you can access from the **Help** menu includes a separate reference guide in HTML format describing the base rules supported by DesignChecker. The relevant HTML pages in the *Base Rule Reference Guide* can also be accessed directly from the corresponding base rule.

A tutorial is available in HTML and PDF format from the DesignChecker InfoHub; to access the tutorial, select **Help and Manuals** from the **Help** menu and then open the **Support and Training** tab.

Additionally, HTML pages concerning **Common Base Rule Parameters** and the syntax of **Regular Expressions** are also available from the **Help** menu.

DesignChecker is an add-on tool; it integrates with various interface tools such as HDL Designer Series, Visual Elite, and so on. You can refer to the integration documents available through the **Help** menu of the interface tool.

It should be noted that the screen shots and examples used in this user guide are taken from DesignChecker using HDL Designer Series as the interface tool. Hence, the content of the screen shots may differ with other interface tools.

For general information about HDL coding practices and test issues, we recommend Michael Keating's and Pierre Bricaud's *Reuse Methodology Manual*.

Related Publications

Michael Keating and Pierre Bricaud, *Reuse Methodology Manual*, 3rd edition, Kluwer Academic Publishers, 2002.

Introduction

The DesignChecker static checking and analysis tool ensures interoperability and consistency across design styles and coding practices for developers and development teams creating complex designs. It is a powerful solution for individual engineers, and can be used in situations where a number of individuals and/or teams are involved in the design and development processes.

DesignChecker can operate in batch mode, or with a full featured Graphical User Interface (GUI). The batch process and commands differ according to the interface tool that is being used with DesignChecker; refer to the integration documents available through the **Help** menu of the interface tool. The user interface supports quick and easy rule configuration, with powerful results cross-referencing between DesignChecker and the editor of the interface tool. For example, in case of interfacing with HDL Designer Series, you will be able to cross-reference between DesignChecker results and the DesignPad text editor and any of the HDL Designer graphical editing tools.

DesignChecker includes a set of parameterized base rules which you use as the building blocks to create the checks you need to run. Base rules are used to create rulesets. The rulesets, which are created by the project manager or team leader, provide the core interoperability and consistency checks, and should be configured to meet appropriate development design and coding standards in the organization. Further information about how to configure rulesets can be found in "Working with Rulesets" on page 40.

You can adapt DesignChecker to meet individual user needs by referencing rulesets and enabling or disabling certain checks through user-defined policies. DesignChecker policies can be configured by individual users to run the subset of checks appropriate to the design. These can be run during coding to provide interactive analysis and feedback, enabling you to rapidly

identify problems during the development cycle. Further information about how to configure and use policies can be found in "Working with Policies" on page 54.

DesignChecker provides powerful, configurable reporting options, enabling you to rapidly identify potential design, simulation and synthesis issues early in the development cycle. The user interface provides powerful results visualization and cross-referencing between the results and the source files. The viewpoint manager includes sophisticated filtering capabilities enabling you to focus your analysis on key areas. Your results can be exported from the GUI as comma separated values (CSV), tab separated values (TSV) or HTML files for ease of reference. Results from running DesignChecker in batch mode can be output to the screen, text or HTML files.

The base rules support VHDL, Verilog, SystemVerilog, or mixed VHDL and Verilog language designs. Definitions and configuration options can be found in the **Base Rule Reference Guide** which you can access from the **Help** menu.

Base Rules

A wide range of parameterized base rules are provided in DesignChecker. Read-only versions of the base rules are stored in the *Base Rules* folder. You can create writable copies of the read-only base rules in your rulesets and policies. You can modify the parameters of these copies. By providing base rules in a configurable and extensible form, you have the flexibility to tailor these to your precise requirements in a simple and intuitive manner. You can find full details of the parameters and configuration options for each base rule in the **Base Rule Reference Guide** in the **Help** menu. Further information about configuring and working with your rulesets and user policies can be found in "Working with Rulesets" on page 40 and "Working with Policies" on page 54.

Note

Base rules that support SystemVerilog are marked with a SystemVerilog overlay  .

Rulesets

A project manager typically sets up one or more rulesets. These will be company or development team specific. Such rulesets provide the core interoperability and consistency checks, and should be configured to meet appropriate development design and coding standards. Further information about how to configure rulesets can be found in "Working with Rulesets" on page 40.

A ruleset can reference other rulesets and configured base rules. DesignChecker comes preconfigured with a number of rulesets that differ according to the interface tool you are using.

For example, in case of HDL Designer Series, DesignChecker comes provided with the Reuse Methodology Manual (RMM) ruleset which is derived from Michael Keating and Pierre

Bricaud's *Reuse Methodology Manual*. In addition, DesignChecker provides the Essentials, Altera, Xilinx, Checklist, Safety-Critical and DO-254 rulesets.

Policies

Any number of rulesets can be referenced in a user defined policy. A policy also contains optional disable settings for the rules referenced by the policy. These settings are specific to each policy. Further information about configuring and working with your user policies can be found in "Working with Policies" on page 54.

Rulesets and Rule Categories

Rulesets are the building blocks of the DesignChecker checking process. DesignChecker supports the following rule categories:

- Standard Language Syntax & Semantics: Specifies the language or dialects that apply.
- Coding Practices: Design style (for example, synchronous or asynchronous, clocking scheme, gated clocks), language constructs and naming conventions.
- Simulation: Limited checking for design style issues that impact simulation (for example, race conditions).
- Synthesis: Static checks that would identify constructs which are typically not synthesizable (for example, clocks, resets and registers) or other synthesis coding issues.
- Testability: Checks for coding styles (for example, combinational loops) that impact design-for-test (DFT).
- Portability, Reuse and Cross-language compatibility: Checks which limit the constructs used in either language to a subset which can reliably be translated between them. These checks are important for re-usable components which need to support either language. These checks can also ensure that designs use a subset which is common to all tools used on the project.

The syntax and semantic checks are a function of each language and therefore common to all designers using that language.

The cross-language compatibility checks are typically also fixed since they relate to constructs which are equivalent between the different languages and dialects.

Most of the other checks are based on generally accepted concepts. However, the exact details and the applicability of rules may vary between organizations, projects and the downstream tools being used. For example, it is generally helpful to have a naming convention to easily identify clocks and resets although the exact name and usage of a prefix or a suffix may vary. Rulesets are used for this purpose, and will normally have been configured to reflect company specific design and coding practices. Individual users can reference rulesets in user policies, and

adapt these to meet specific needs. For further information about configuring and working with user policies refer to "Working with Policies" on page 54.

Design Checking Flow

The DesignChecker flow typically follows this process:

1. Choose an existing policy or create a new policy. A policy references rulesets and specifies which rules are enabled or disabled.

   Rulesets consist of configured base rules, and should be created beforehand by a project manager or team leader.

2. Specify which files or objects to include or exclude from analysis.
3. Run the analysis.
4. View the results. The results can be sorted, grouped, filtered, summarized and exported.
5. Cross-reference error and warning messages to the source text, or graphical diagram or table source views.

Checking Mixed Language Designs

DesignChecker can check files written in either VHDL or Verilog.

If the design being analyzed is single language and a rule is enabled for this language, the rule will be applied to all the files.

If the design being analyzed is single language and a rule is disabled for this language, it will not be applied to any of the files.

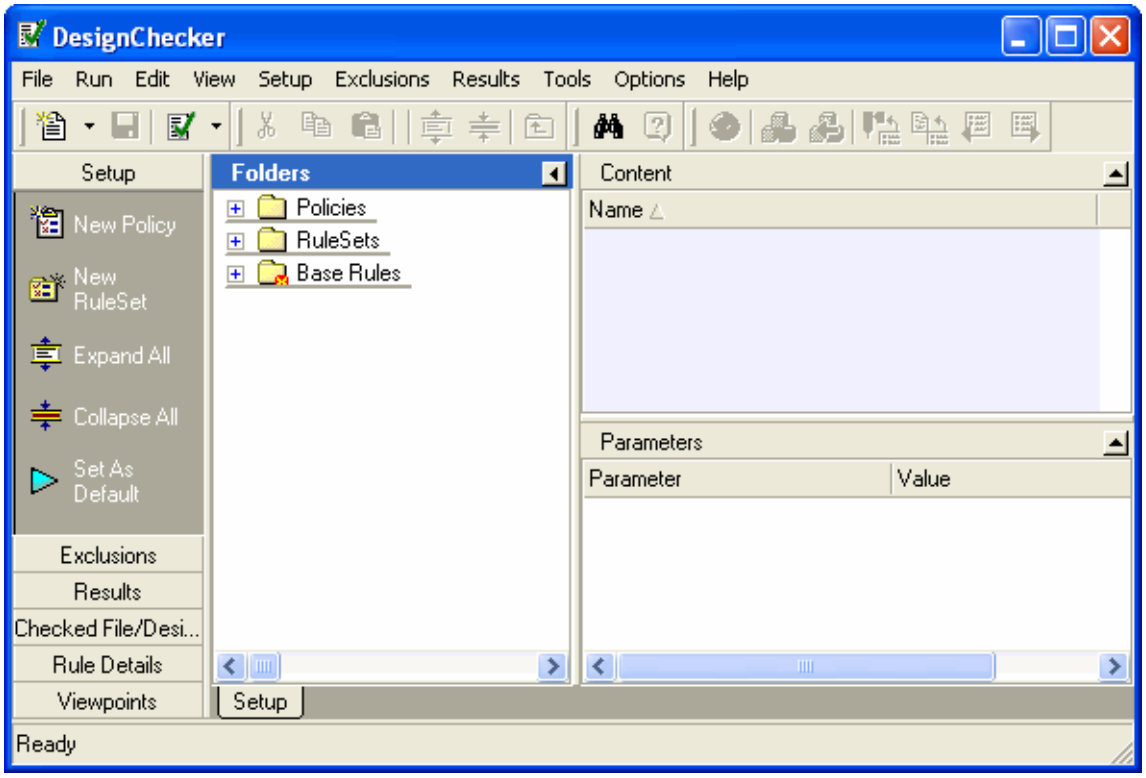
If the design being analyzed is mixed language, a rule will run for all the files regardless of its language settings as some errors or associated files could be in an HDL file whose language differs from the default language setting.

Invoking DesignChecker

DesignChecker can be invoked through the interface tool. Refer to the integration documents available through the **Help** menu of the interface tool.

For example, in case of HDL Designer Series, you can invoke DesignChecker by choosing Manage Policies/Rulesets from the Tools menu in the main HDL Designer Series design manager.

Once DesignChecker has been invoked a separate DesignChecker window is launched with the Setup tab active.



You use the Setup tab to configure your rulesets and policies. A ruleset contains the full suite of design checks that can be run. A policy determines which of these checks will actually be run, and specifies which rules should be excluded from analysis. Both rulesets and policies can be configured to meet your needs.

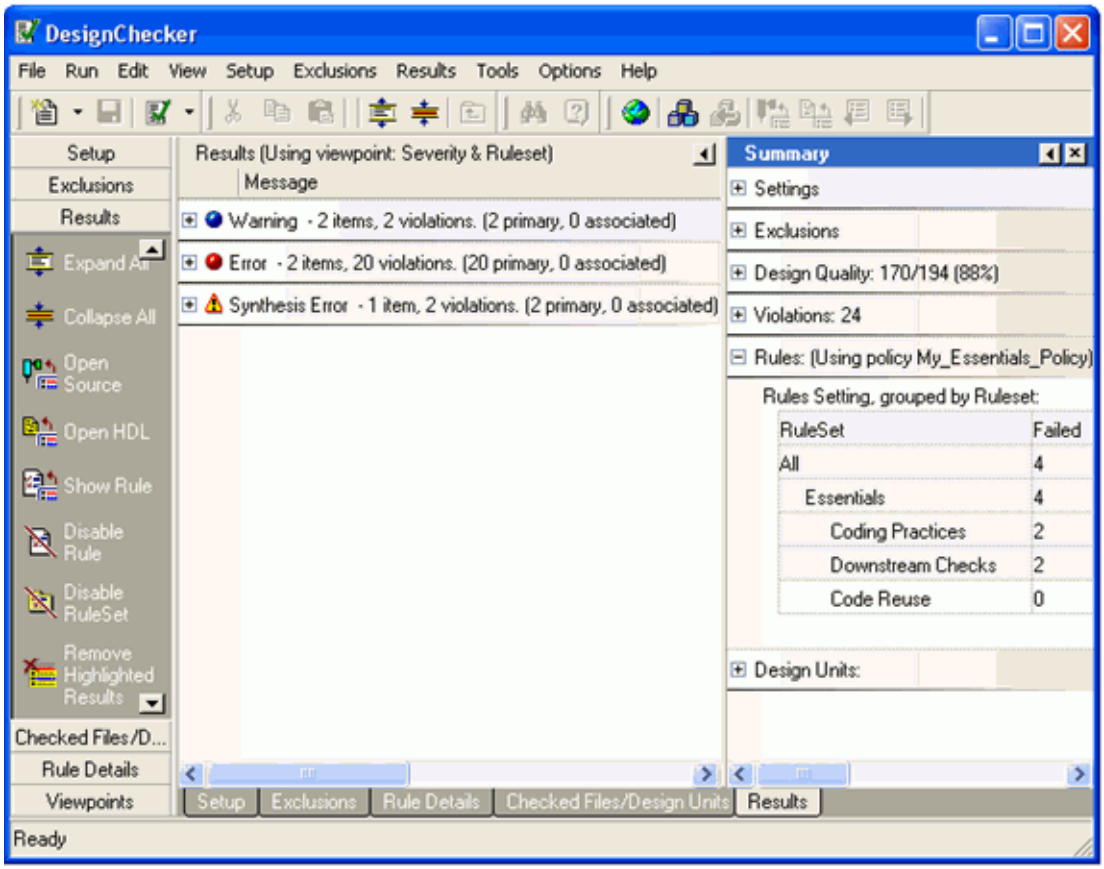
Refer to "The Setup Tab" on page 19 for further information on how you use the Setup tab.

Running DesignChecker

You can run DesignChecker directly through the interface tool. Refer to the integration documents available through the **Help** menu of the interface tool.

For example, if you are using DesignChecker with HDL Designer Series, then you can invoke it by running the default analysis on a specified file/design unit by using the  button or by choosing **DesignChecker Flow** from the **Tasks** menu in the design explorer, a graphical editor or the DesignPad text editor. Consequently, the DesignChecker flow generates HDL for the specified view or hierarchy of views (if the selection includes graphical views) and runs the default design checks policy.

DesignChecker is invoked with the current default viewpoint displayed in the Results tab. For example the following picture shows the results displayed in the *Severity & Ruleset* viewpoint after an analysis has been performed.



The  button and menu provide options to run the analysis. For example, in case of HDL Designer Series, you can run the analysis on a single level, through blocks or components from the selected object, or through all components below the design root.

In DesignChecker you can choose **Re-run Last Analysis** from the **Run** menu to re-run the last analysis.

Refer to "The Results Tab" on page 68 for further information about interacting with the results.

Note

The DesignChecker Exclusions tab displays the current exclusion settings. For example, in case of HDL Designer Series, this tab shows the code/rule exclusions, black box exclusions, pragmas, and so on. Exclusions allow you to ignore specific parts of the design or specific rules during the analysis. For further information, refer to "Setting Exclusions" on page 64.

Note

It should be noted that for some checks to run properly in DesignChecker, the designs must be complete and synthesizable first. This is applicable to specific rules in DesignChecker; refer to "Rules Specific to Synthesizable Code" on page 101.

Exiting from DesignChecker

You can exit from DesignChecker by choosing **Exit** from the **File** menu. You are prompted to save any rulesets or policies that have not been saved.

The position, size and window layout (including the active viewpoint used for the last analysis) is remembered and re-used when DesignChecker is next invoked.

Chapter 2
Configuring DesignChecker

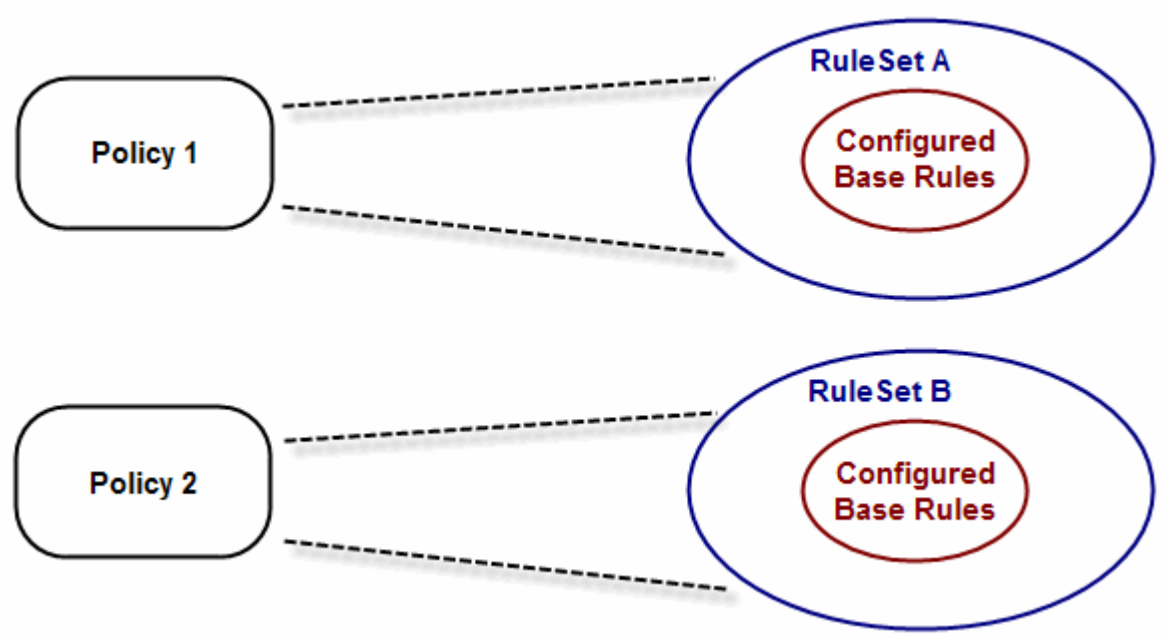
This chapter describes the configuration options for DesignChecker. It explains how to configure your analysis options using the parametrized base rules, how you can selectively implement checks with user policies, how you can assess the quality of your design, and so on.

Introduction

DesignChecker provides a flexible and intuitive analysis and static checking solution. You use rulesets and policies in conjunction to define the precise checks carried out each time you run DesignChecker. Your organization will implement their design and coding standards in rulesets. You can use your policies to control which checks are applied to each DesignChecker pass. This enables you to focus your attention on specific design and development issues as work progresses, ensuring that you are presented with results that are relevant to your immediate development needs, rather than being overwhelmed with information.

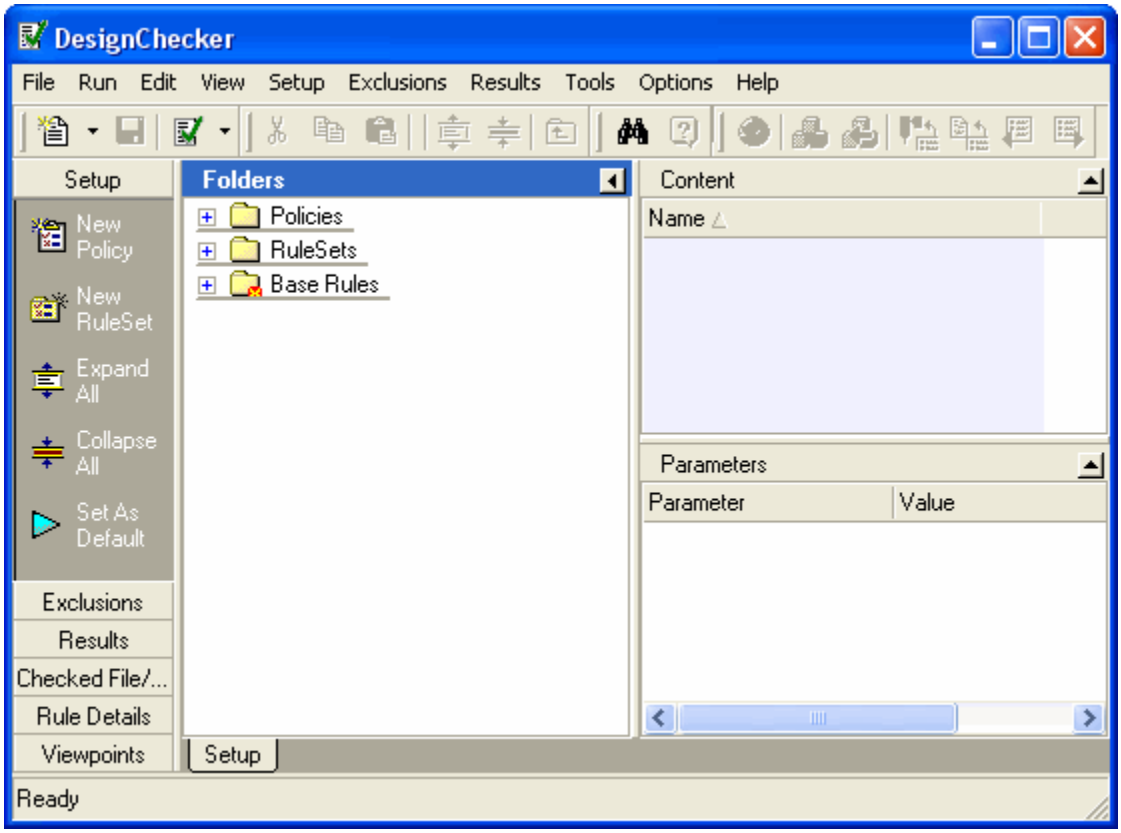
That is to say, the project manager or team lead is basically responsible for configuring rulesets. Policies, the entity that references rulesets and is applied to your analysis, can be created by team members (or by the project manager or team lead at times). While running an analysis, team members can use the policy to control which rules should or should not be applied in that specific analysis.

For example, the following figure shows RuleSet A and RuleSet B. The project manager creates these rulesets using the variety of configurable base rules provided by DesignChecker (according to the coding standards of the organization). Team members create policies to reference the rulesets, and they select a default policy to use when running the analysis. Moreover, when running the analysis, team members have the ability to disable certain rules only on the policy level (if needed).



Initial Settings

When you invoke DesignChecker for the first time, the base rules are loaded in addition to built-in rulesets and user policies. The setup tab displays three folders for *Base Rules*, *RuleSets* and *Policies*:



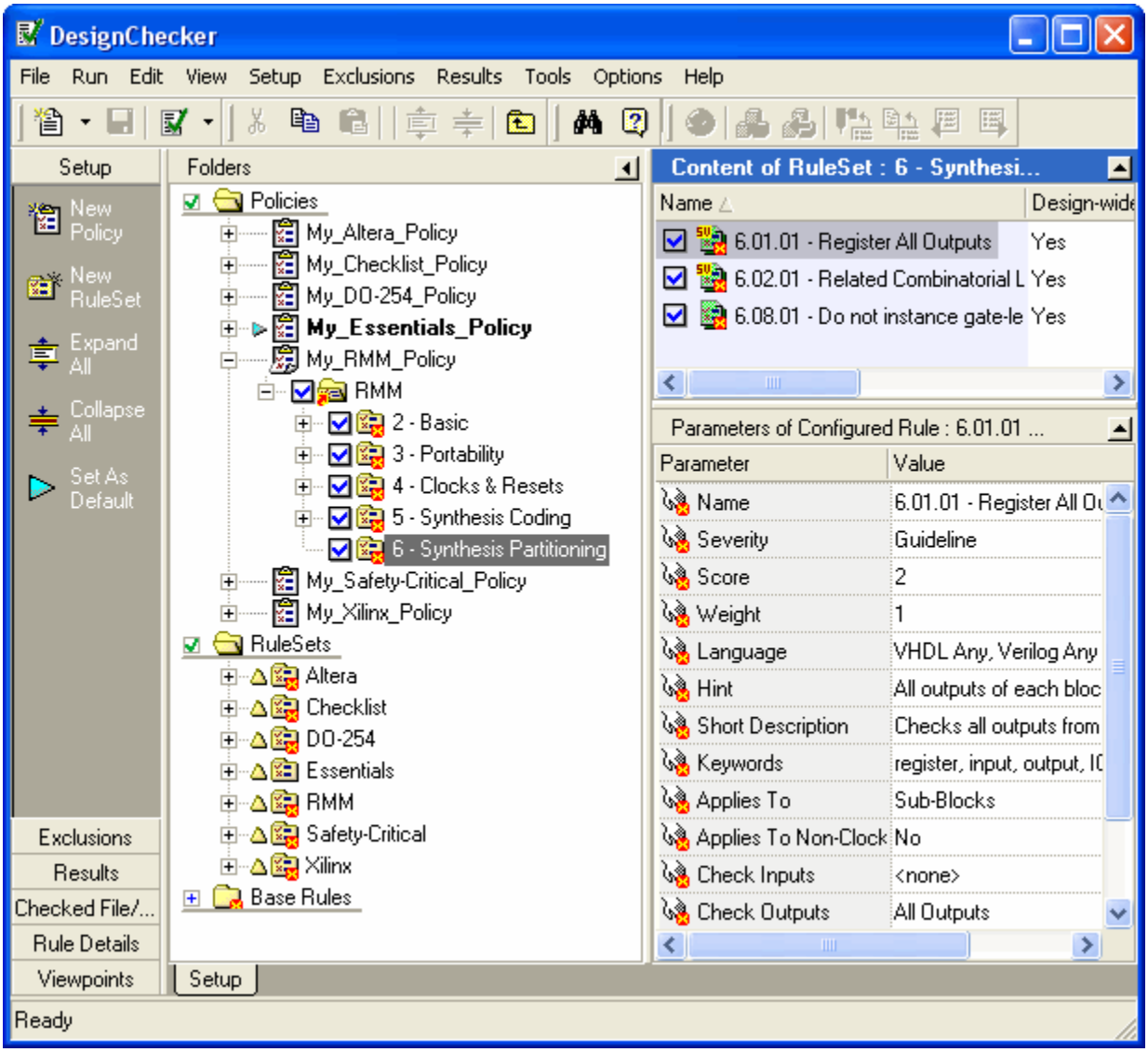
Note

The *Base Rules* folder has an overlay indicating that the base rules are read-only. The *RuleSets* and *Policies* folders you create are initially shown with an overlay. This indicates that they have not yet been saved. All new or edited rulesets and policies are shown as modified until they have been saved.

The Setup Tab

You use the setup tab to configure your DesignChecker rulesets and policies. The setup tab opens automatically when DesignChecker is launched.

There are three main regions - the Folders pane, the Content pane and the Parameters pane. You use these areas in conjunction to create and adapt rulesets and policies to meet your development needs.



The Folders pane contains folders for *Base Rules*, *RuleSets* and *Policies*. The *Base Rules* folder provides you with the building blocks for creating rulesets and policies appropriate to your organization's and your own development requirements.

If you are working as a member of a development team, team specific rulesets should be available to you within the *RuleSets* Folder. Rulesets are normally created by a project leader and are read-only for team members. Details about how to set up rulesets for project leaders can be found in "Creating a Ruleset" on page 43.

Team members can adapt rulesets using individual user policies. These enable you to turn on or off selected sets of checks as well as adapting the results and the reporting options to best suit

your needs. You can create as many policies as you need. You create your policies in the *Policies* folder. Details about how to set up individual user policies can be found in "Working with Policies" on page 54.

Setup Tab Notation

The setup tab displays information using the icons shown in the following table:

Table 2-1. Setup Tab Notation

Icon	Description
	Base rule folder
	Open base rule folder
	Policy folder
	Open policy folder
	Ruleset folder
	Open ruleset folder
	Base rule
	Configured rule
	Parameter
	Default policy marker
	Enabled rule or ruleset
	Partially enabled ruleset
	Disabled rule or ruleset

Viewing the Base Rules

Base rules are provided for the following categories:

Allow	Labels
Assignments	Logic Optimization
Case	Naming
Clocks & Resets	Order
Comments	Partition
Complexity	Race Conditions
Conditions	Ranges
Configurations	Registers
Dead Logic	Report
Declarations	Sensitivity
Directives	Style
FSM	Sub-programs
Gates	VITAL
Hierarchy	
Instances	

Each base rule relates to a single topic and has a number of configurable parameters.

The base rules are read-only. You can configure them by making a copy and adapting the copy's parameters to make a ruleset. A wide variety of specific checks can be produced using base rules in this way, enabling you to adapt, extend and apply your rulesets and policies in an intuitive and flexible manner.

You can view the base rules by expanding the *Base Rules* folder and selecting the required rule. You can search for appropriate base rules using keywords in the search field or by using the **Base Rule Reference Guide** from the **Help** menu. Refer to "Using the Search Bar" on page 26 for further information.

Note

To use DesignChecker effectively you must identify and adapt the most appropriate base rules for your purposes. The use of parameterized base rules in DesignChecker means that there may be multiple ways to implement a specific check. Care should be taken to ensure that your rulesets and policies accurately reflect your design and coding standards. Refer to "Working with Rulesets" on page 40 and "Working with Policies" on page 54 for further information on how to configure these options.

Note

Base rules that support SystemVerilog are marked with a SystemVerilog overlay  in the Content pane.

The content of the base rule category is shown in the Content pane. On selecting any base rule category folder in the Folders pane, the Content pane consequently displays all the rules belonging to that category along with data about each rule. This data is presented in the form of columns as follows:

- **Base Rule Category** — The name of the category to which the base rules belongs.
- **Base Rule Name** — The name of the base rule.
- **Design-wide** — This column indicates whether the base rule can be configured and applied to the entire design hierarchy (Yes) or not (No). This column is applicable with certain interface tools such as HDL Designer Series and Visual Elite.

  It is important to note that a design-wide rule depends on your configuration of the rule. That is to say, if a rule is indicated as being design-wide (Yes), then this does not necessarily mean that the rule already has a design-wide configuration that is suitable for running on a full design; this rather means that the rule has capacity for such configurations.

  Additionally, if you are running a DesignChecker analysis using design-wide rules, you should pay attention to the level of analysis you wish to run according to the object of your analysis (that is, whether the analysis includes a full design or only a part of it). This affects the accuracy of your results.

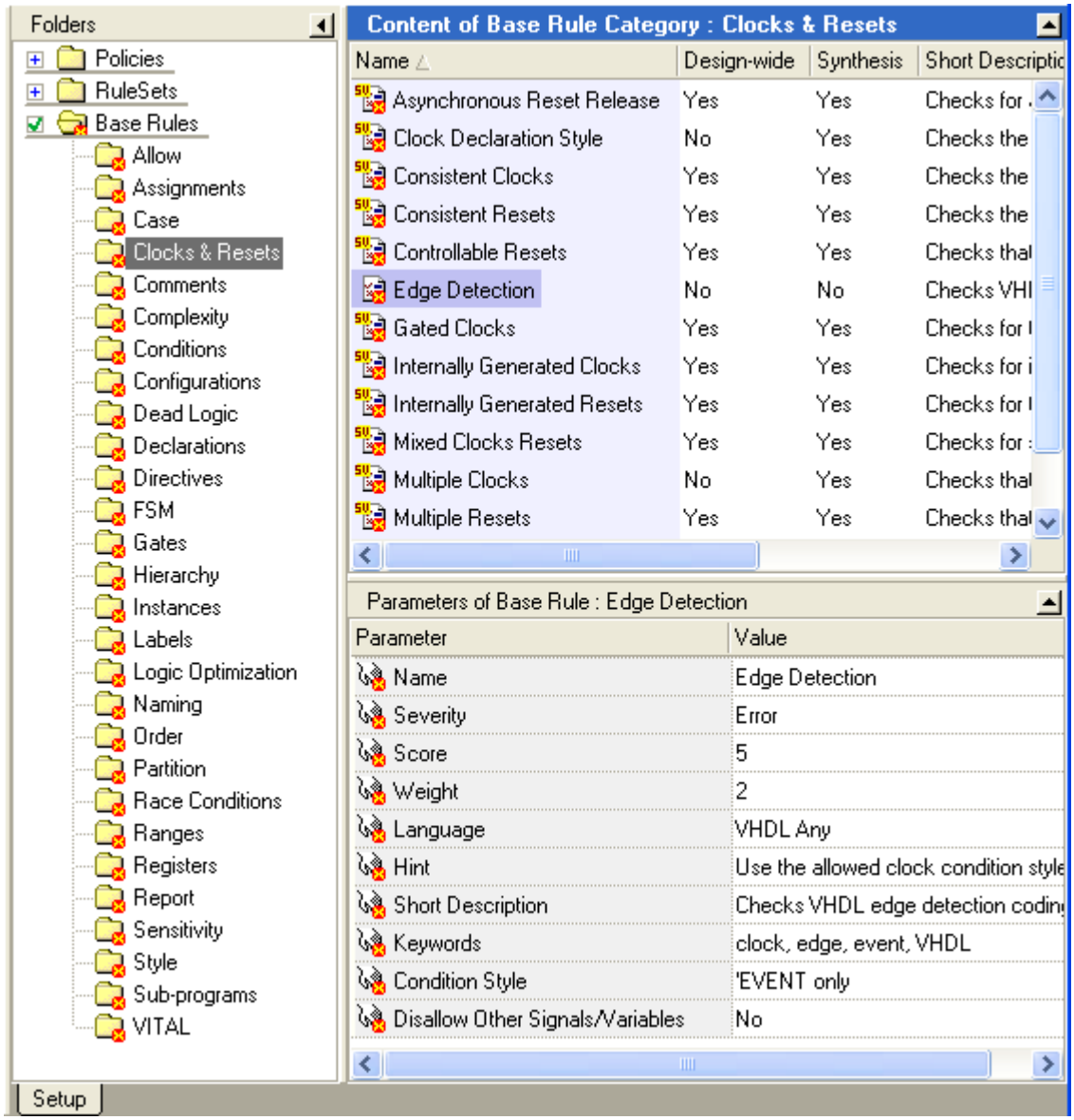
  For example, if you want to analyze an individual design unit, running a single-level analysis may not be the best choice if this design unit is dependent on other design units. This is because the chosen design unit will not solely include all the information required by the design-wide rules, thus leading to inaccurate results. In that case, you should select a different depth for analysis such as Through Components or Through Design Root.

  As previously mentioned, you can run the analysis on a single level, through blocks or components from the selected object, or through all components below the design root.
- **Name** — The name by which the base rule is identified.
- **Short Description** — A brief note on the function of the base rule.
- **Synthesis** — This column indicates whether the rule can run only on synthesizable code or not. Refer to "Rules Specific to Synthesizable Code" on page 101. This column is applicable with certain interface tools, mainly with RTL tools such as HDL Designer Series and Visual Elite.
- **Type** — The type of the rule (base rule, ruleset, or configured rule). In this case (that is, in case of viewing the base rules of a category), this column always displays the type as Base Rule.

Only the following columns are displayed in the Content pane by default: **Name**, **Design-wide**, **Synthesis**, and **Short Description**.You can control which information to display in the Content pane by doing the following:

1. Right-click on the title bar of any column in the Content pane.
2. From the **Select Columns** cascade, choose whether you want to display **All Columns**, **No Columns**, or specific columns such as **Base Rule Category**, **Base Rule Name**, **Design-wide**, **Name**, **Short Description**, **Synthesis**, or **Type**.

When you select a rule in the Content pane, its default parameters are shown in the Parameters pane. The following example shows the parameters for the *Edge Detection* rule in the *Clocks & Resets* category:

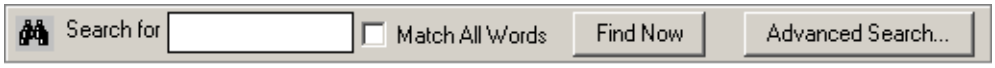


You can access detailed information and configuration options for each base rule by choosing **Base Rule Reference Guide** from the **Help** menu to display the *Base Rule Reference Guide* in your HTML browser.

You can directly access the description of a base rule by using the  button or by choosing **Base Rule Details** from the popup menu in the Content pane when the required rule is selected.

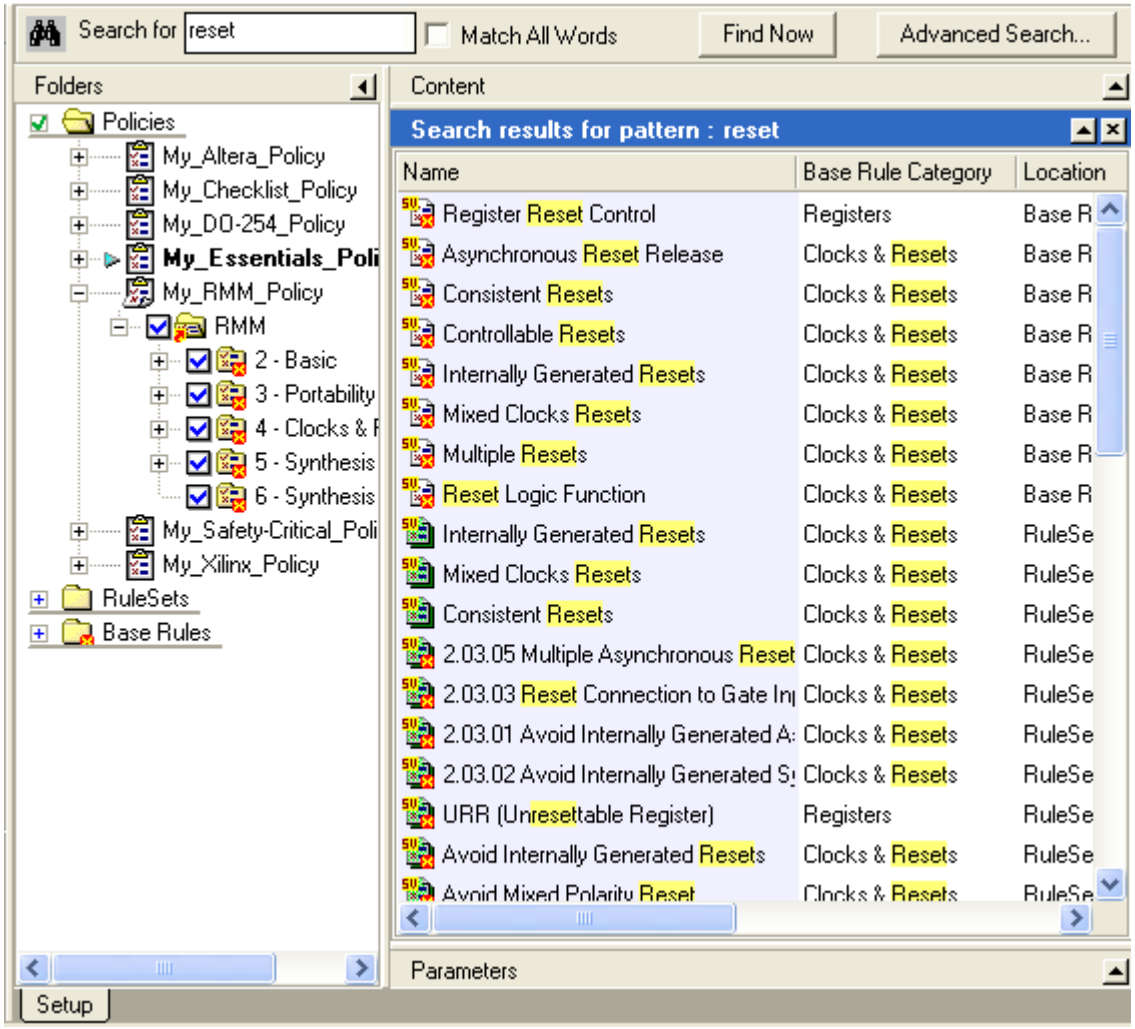
Using the Search Bar

Identifying the appropriate base rule or ruleset is the key to configuring your own rulesets and policies. You can search for a text string within any rule name or description by using the button to display the Search bar:



The search is performed when you click the Enter , Return or Find Now key and any rules that match the search strings are displayed in a new *Search* pane between the *Content* and *Parameters* panes.

You can enter one or more search strings and choose whether to match all words. For example, the following rules are found when you search for the string *reset*:



If you enter multiple strings, all matching rules will be found. For example, if you enter the strings *reset control*, all rules containing *reset* or *control* are found. However, if you set the **Match All Words** option, only the rules containing both strings together will be found.

For a more refined search, click **Advanced Search** to customize your own search options. Refer to "Using the Advanced Search" on page 27, for further information.

Using the Advanced Search

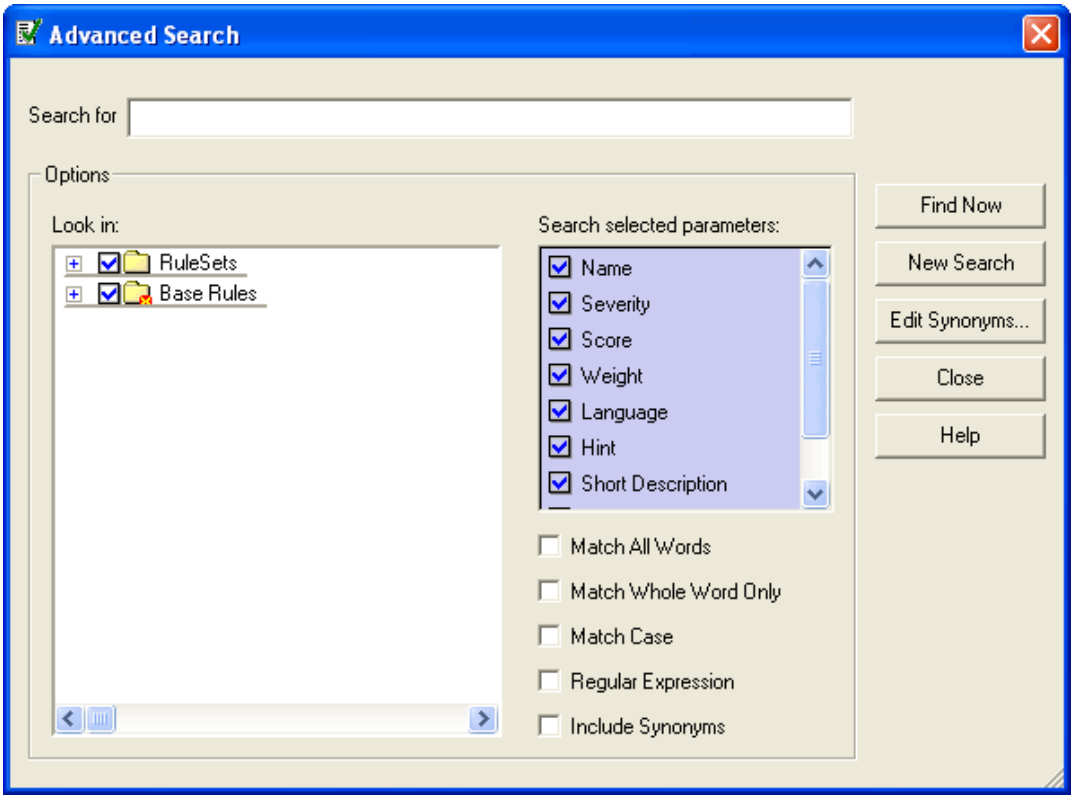
In order to create your own rulesets, DesignChecker enables you to search for the corresponding base rule using two methods: the Simple Search, and the Advanced Search. Whereas the simple search automatically looks for the search string only in the Name and Short Description of the available base rules and configured rules, the advanced search gives you the ability to expand or limit the scope of your search to ensure more specific results. Refer to "Using the Search Bar" on page 26, for further information on simple searches.

Running an Advanced Search

Advanced searches enable you to customize your search options to facilitate the retrieval of the required results.

To run an Advanced Search:

1. Open the Setup tab. From the **Tools** menu, select **Advanced Search**. Another method to invoke the Advanced Search dialog is through the **Advanced Search** button on the simple search bar.
2. Click **New Search** to initialize the dialog.



3. Enter the search string in the “Search for” field.
4. Specify an exact search location using the “Look in” section, where a hierarchical view of the available rulesets and base rules is displayed. Expand the tree and select the check box of the required ruleset or base rule category in which the search shall occur. Note that all rules are selected by default.
5. Select the required search parameters in the Search Selected Parameters section. By default, the simple search looks for the string in the Name and Short Description parameters only. Unlike the simple search, the advanced search allows you to specify the parameters to look in.

   The parameters available are Name, Severity, Score, Weight, Language, Hint, Short Description, and Keywords, in addition to Rule-specific Parameters.

   This allows a more flexible search; for instance, you can search for the string *Warning* and select only the parameter *Severity*, this leads to obtaining all rules that have the severity set as warning.

Note

You can manually edit the Keywords parameter of your configured rules to define the terms relevant to those rules. This facilitates future searches for your configured rules. That is to say, the keyword can be typed as a search string to retrieve all configured rules associated to that keyword.

6. Select a search option if necessary. The available search options are as follows:
   - Match All Words — This option obtains the rules that contain all the entered strings, regardless of the word order. For example, if you enter *clock reset*, all rules including both strings *clock* and *reset* are retrieved such as the base rule *Mixed Clocks Resets*.

Note

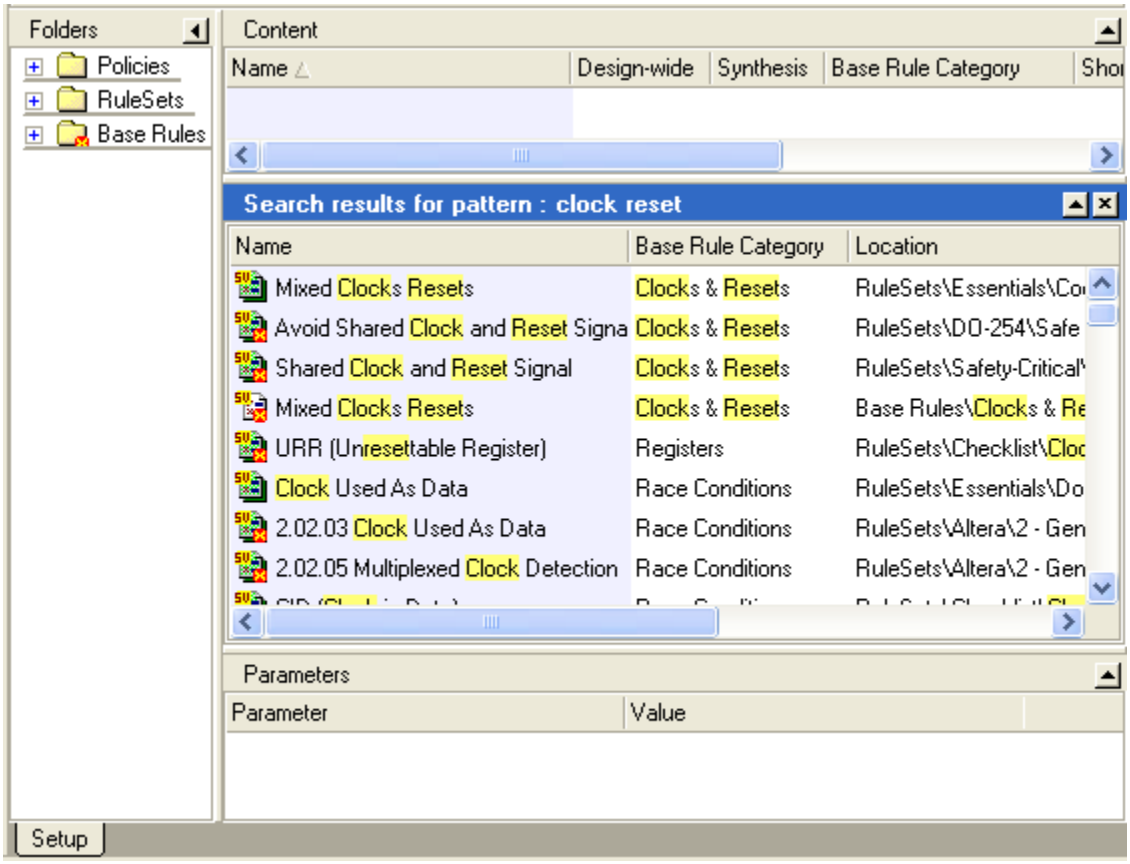
To search for multiple strings in their exact order, enclose the strings between double quotations. For instance, type *“gated clocks”* and select Match All Words, on clicking **Find Now**, you will receive only rules that have both strings in the same entered order.

   - Match Whole Word Only — This option obtains the rules that contain the search string exactly as it has been entered. For example, if you type *clock* and select Match Whole Word Only, then you will receive only rules having the word *clock* - such as the Clock Boundaries rule - discarding any other form of the word such as *clocks*, *clocked*, and so on.
   - Match Case — This option allows a case-sensitive search; it obtains the rules that match the search string in the exact entered case.
   - Regular Expression — This option enables you to search for strings using regular expressions. For example, entering the search string \<*cl* would find strings starting

with the characters *cl*. A list of supported regular expressions can be displayed by choosing **Regular Expressions** from the **Help** menu.

- Include Synonyms — This option allows you to search for synonyms of the entered string. For more details, refer to "Running an Advanced Search with Synonyms" on page 33.

7. Click **Find Now**; consequently, the search results are displayed in the Search pane. You can manipulate the resulting rules through the Search pane directly; for example, you can right-click on the required rule and select **Copy** from the popup menu, and then right-click on the relevant ruleset (in the Folders pane) and select **Paste**.



8. Click **Close** to exit the Advanced Search dialog.

Note

The Advanced Search dialog preserves your last search entries; in other words, on closing and re-opening the Advanced Search dialog, you will find the last entered search string and search options. To initialize the dialog, click **New Search**.

Other Methods to Run an Advanced Search

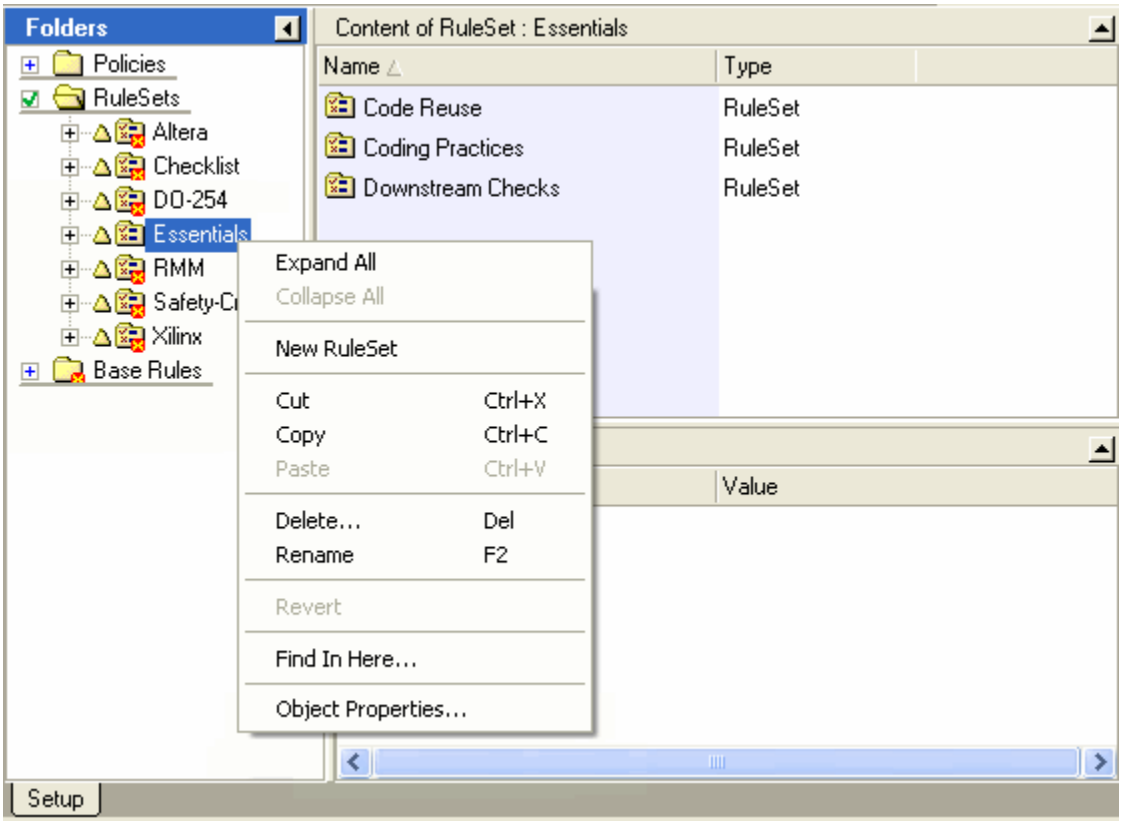
There are two more methods to run an advanced search on rules: Find In Here and Find Where Used.

Find In Here

This method enables you to invoke the Advanced Search dialog box directly through the ruleset or base rule folder in which you want the search to take place.

To search in a RuleSet or Base Rule Category:

1. In the Folders pane, right-click on the ruleset or base rule category.



   Or you can right-click on a ruleset in the Content pane.

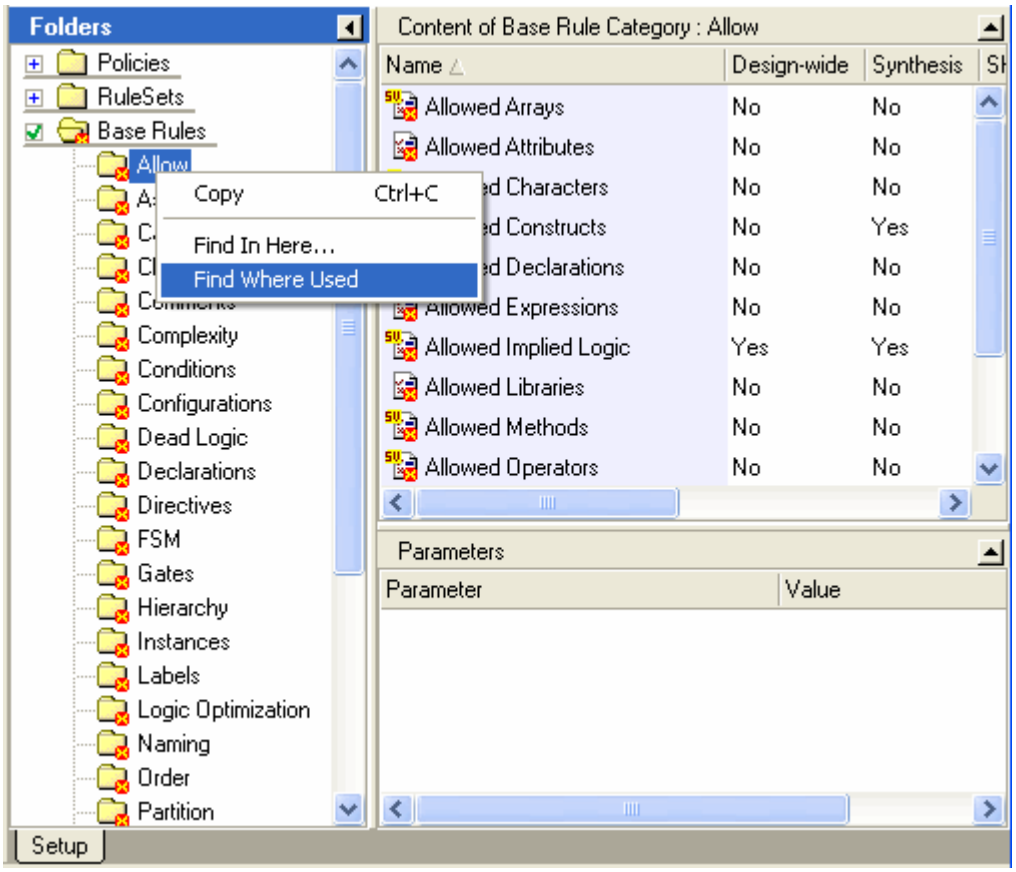
2. Select **Find In Here**; consequently, the Advanced Search dialog opens. Note that the search location is automatically specified in the “Look in” section.
3. Enter the search string and select the necessary options.
4. Click **Find Now**.

Find Where Used

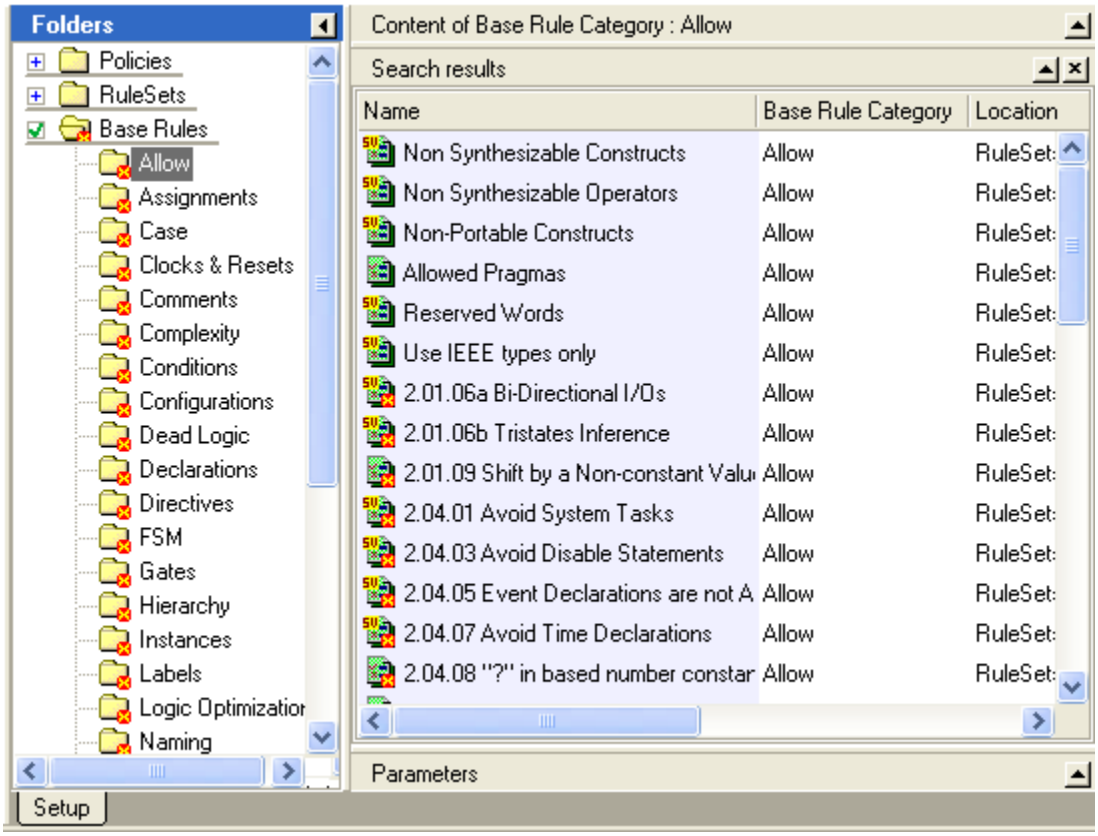
This feature enables you to identify the rules in which a base rule has been previously configured. This helps you to avoid creating the same rules for the same base rule more than once.

To find where a Base Rule is used:

1. In the Folders pane, right-click on the required base rule category; or you can right-click on a specific base rule in the Content pane.



2. Select **Find Where Used**. Therefore, all the configured rules using the base rule appear in the search results in the Search pane.



Viewing Search Results

After running a search, the results are displayed in the Search pane. The search results section is characterized by the following:

- Highlighting Search Strings: The values corresponding to the specified search string are highlighted in yellow.
- Sorting Search Results: The search results are ordered from the most relevant result - that is to say the result which matches the largest number of search strings - to the least relevant result.

  For instance, if you enter *Gated Clocks* as the search string and click **Find Now**, all rules containing both words *Gated* and *Clocks* appear first, then followed by the rules containing either the word *Gated* or the word *Clocks*.
- Displaying Results in Columns: The results' data are displayed in the form of columns. The default columns are Name, Base Rule Category, Location, and Short Description. The displayed columns can be easily customized.

  To customize results' columns:

a. In the Search pane, right-click on the header row of the search results, and choose **Select Columns**.

b. Select whether you want to display **All Columns**, **No Columns**, or specific columns such as **Base Rule Category**, **Base Rule Name**, **Design-wide**, **Language**, **Location**, **Name**, **Short Description**, **Synthesis** or **Type**.

Note

The Search pane can be closed through the button at the upper right corner.

Running an Advanced Search with Synonyms

Terminologies commonly used in one company may differ from the terminologies used in the base rules existing in DesignChecker. This may act as a hindrance when running a search.

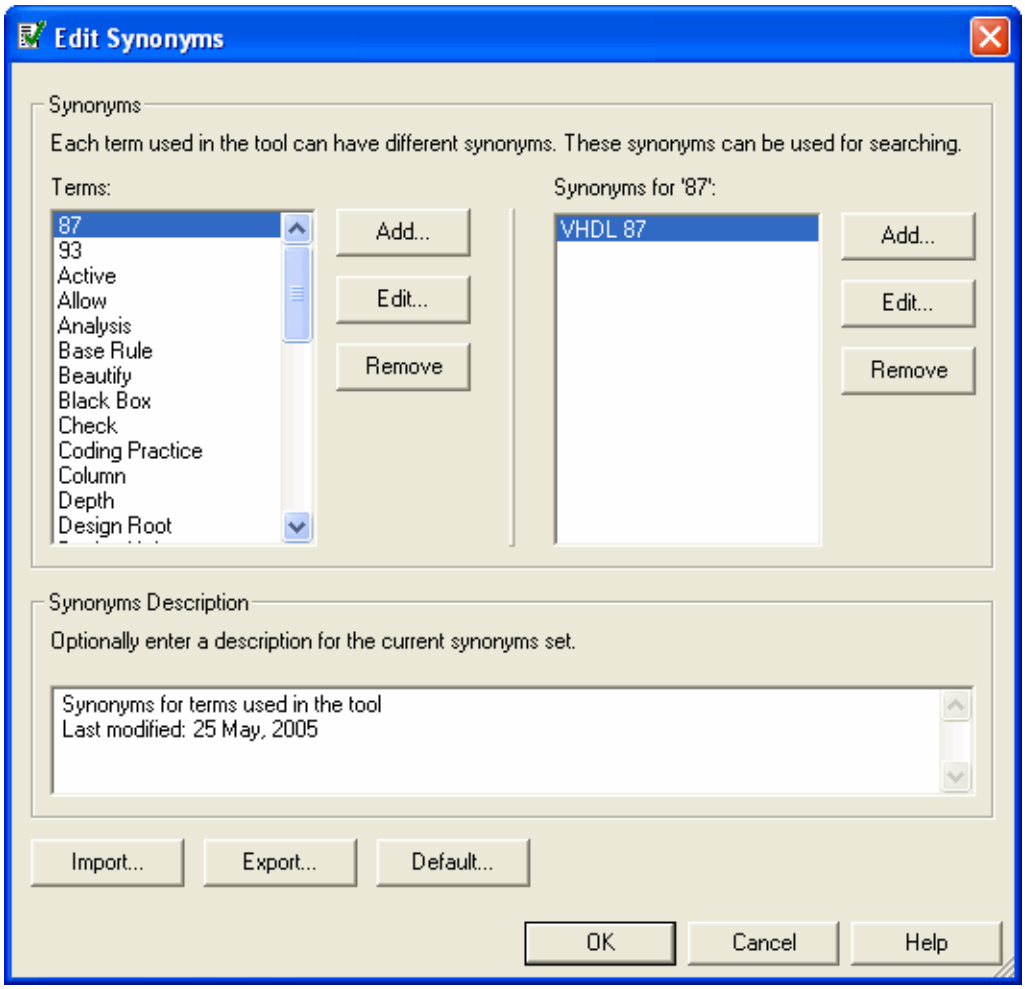
The senior team member can, thus, define the company's synonyms in the DesignChecker, export these synonyms in a text file, and send this file to other team members to import into their systems.

Editing Synonyms

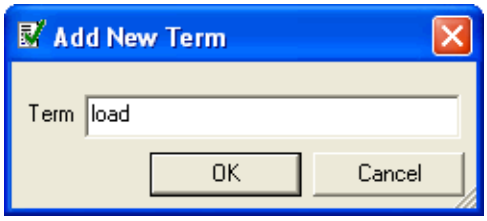
The senior team member defines the company's terms and links them to their corresponding DesignChecker terms. For example, some companies may use the term *control signal* instead of the term *load*.

To edit synonyms:

1. Open the Setup tab. From the **Tools** menu, select **Edit Synonyms**. Another method to invoke the Edit Synonyms dialog is through the **Edit Synonyms** button in Advanced Search dialog.



2. Click **Add** next to the Terms list; in the Add New Term dialog, enter the original term existing in the DesignChecker such as the term *load* and click **OK**.



3. Click **Add** next to the Synonyms list; in the Add New Synonym dialog, enter the equivalent term such as the term *control signal* and click **OK**.

4. Use the Synonyms Description text area to enter general notes on the current list of terms.
5. Click **OK** in the Edit Synonyms dialog box; on re-opening the dialog box later, you will find the term and its synonym added.

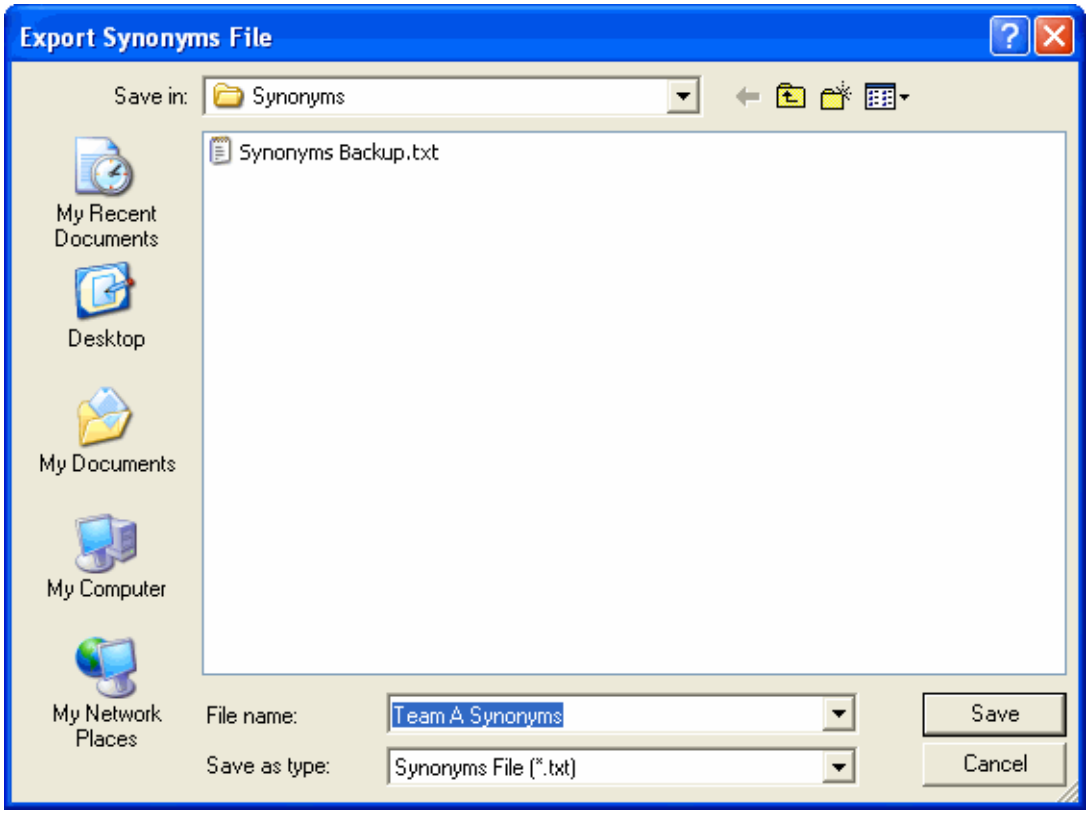
Exporting Synonyms

Having added the required synonyms to the DesignChecker, the senior team member can now export the current existing synonyms into a text file to be sent to other team members, or to be kept as backup.

To export synonyms:

1. Open the Setup tab. From the **Tools** menu, select **Edit Synonyms**. Another method to invoke the Edit Synonyms dialog is through the **Edit Synonyms** button in Advanced Search dialog.
2. Click **Export**.

3. In the Export Synonyms File dialog, specify the path in which the text file shall be kept and enter the File Name of the exported synonyms file.



4. Click **Save**.

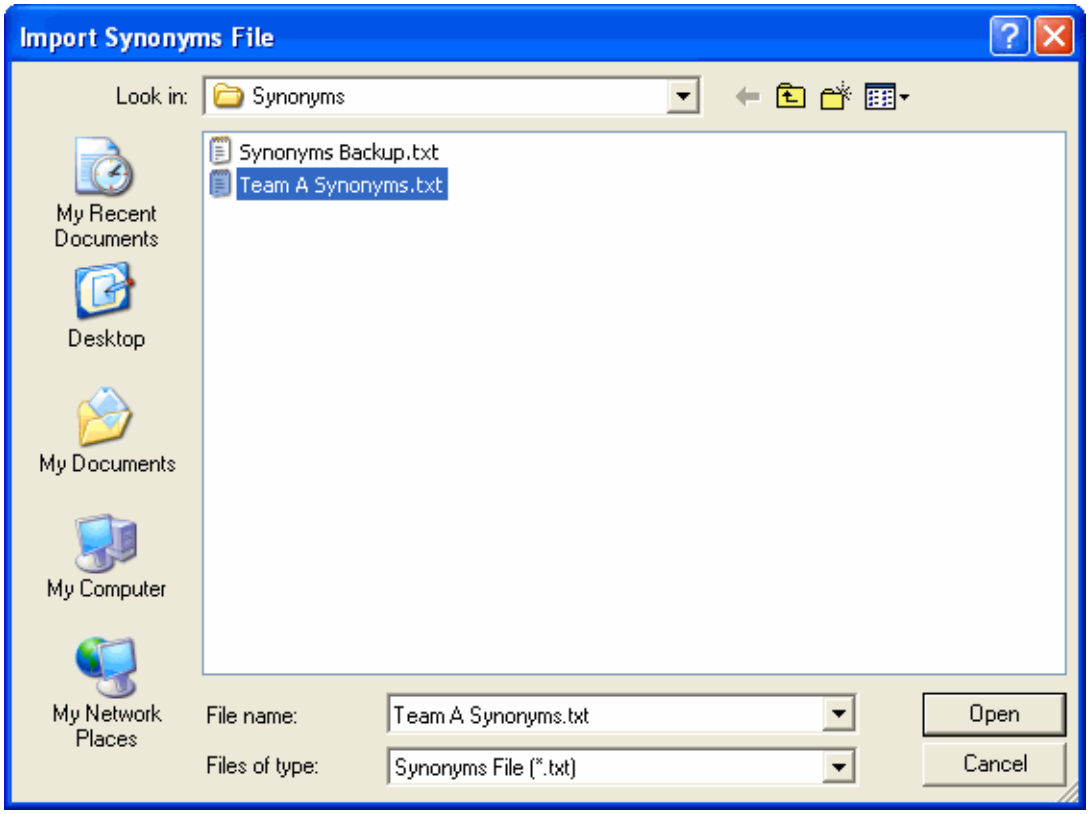
Importing Synonyms

Text files including previously exported synonyms are sent to team members, who in their turn import these text files into their systems.

To import synonyms:

1. Open the Setup tab. From the **Tools** menu, select **Edit Synonyms**. Another method to invoke the Edit Synonyms dialog is through the **Edit Synonyms** button in Advanced Search dialog.
2. Click **Import**.

3. In the Import Synonyms File dialog, browse for the path from which the text file shall be obtained and select the file of the imported synonyms file.



4. Click **Open**.

Restoring Default Synonyms

After adding any synonyms, you may need to discard those added synonyms and restore the default synonyms list existing in DesignChecker.

To restore default synonyms:

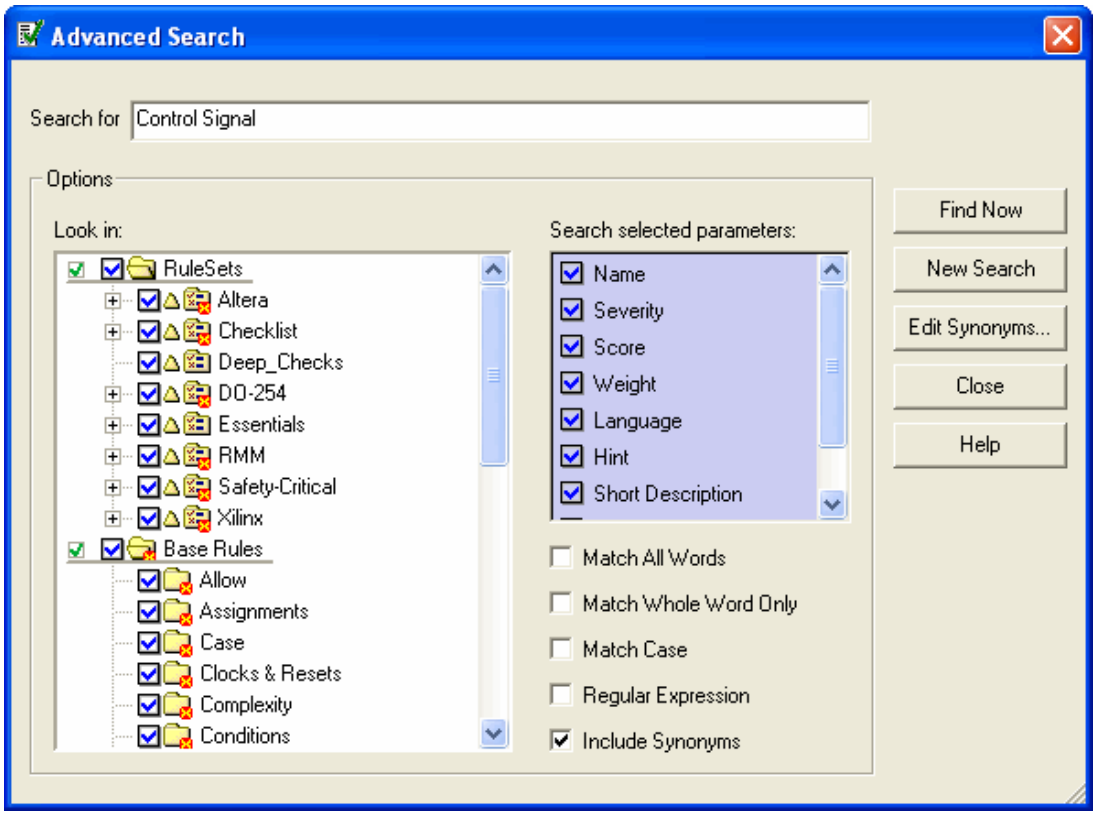
1. Open the Setup tab. From the **Tools** menu, select **Edit Synonyms**. Another method to invoke the Edit Synonyms dialog is through the **Edit Synonyms** button in Advanced Search dialog.
2. Click **Default**.
3. A message is raised requesting your confirmation to replace the current synonyms with the default ones; click **Yes**. By that, any previously added terms and synonyms are deleted from the list.

Applying Advanced Searches Using Synonyms

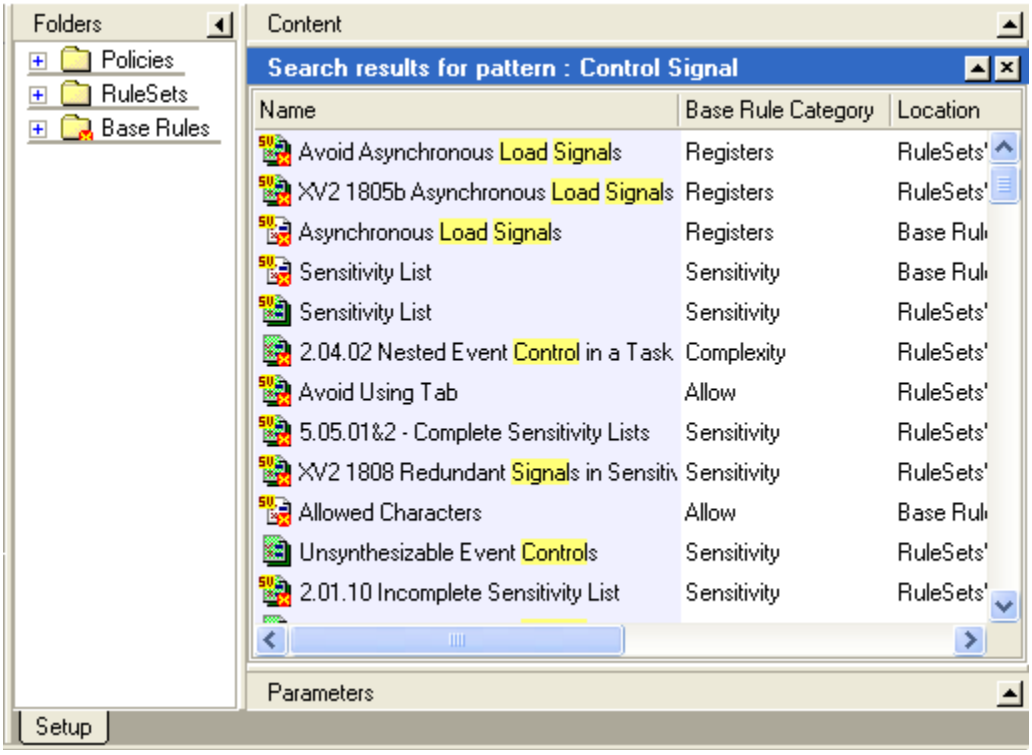
After adding your company's synonyms in the DesignChecker, you now have the ability to search for rules using your own company-specific terms.

To use Advanced Searches with Synonyms:

1. Open the Setup tab. From the **Tools** menu, select **Advanced Search**. Another method to invoke the Advanced Search dialog is through the **Advanced Search** button on the simple search bar.
2. Click **New Search** to initialize the dialog.



3. Enter the synonym in the “Search for” field. For example, if you have defined *Control Signal* as a synonym for *Load*, then enter the search string as *“Control Signal”*.
4. Select the Include Synonyms option.
5. Click **Find Now**; consequently, the results displayed in the Search pane include all the rules having the term *load* such as *Asynchronous Load Signals* and *Operator Overloading*.



Working with Rulesets

A ruleset can contain any number of configured rules and lower level rulesets. Your rulesets should contain the full range of interoperability, design and coding checks used within your organization. A development team leader or project manager normally defines these using the DesignChecker base rules. Such rulesets are specific to your organization and so will vary between different organizations.

Configuring rulesets is normally a ‘once only’ operation, as the rulesets are intended to ensure that team specific design and coding standards are being adhered to - consequently rulesets should be transferable between projects. Rulesets can be set to read-only by specifying file permissions on individual ruleset files. See “Saving Rulesets, Policies and Preferences” on page 57 for information on how to locate your ruleset files.

The team leader can share rulesets by saving them in a single location that can be accessed by all team members. The steps of sharing rulesets differ according to the interface tool you are using. Refer to the integration documents available through the **Help** menu of the interface tool for further details on sharing rulesets.

The default *Essentials* ruleset is provided as a core set of rules typical of those used in the design community. This ruleset is editable as it allows modifications if necessary.

RMM ruleset is preconfigured and, like all rulesets, can be viewed by expanding the *RuleSets* folder. This read-only example ruleset implements the RTL coding guidelines described in the third edition of the *Reuse Methodology Manual*.

The *Altera* and *Xilinx* rulesets are preconfigured to be compatible with the design code implemented using Altera and Xilinx tools.

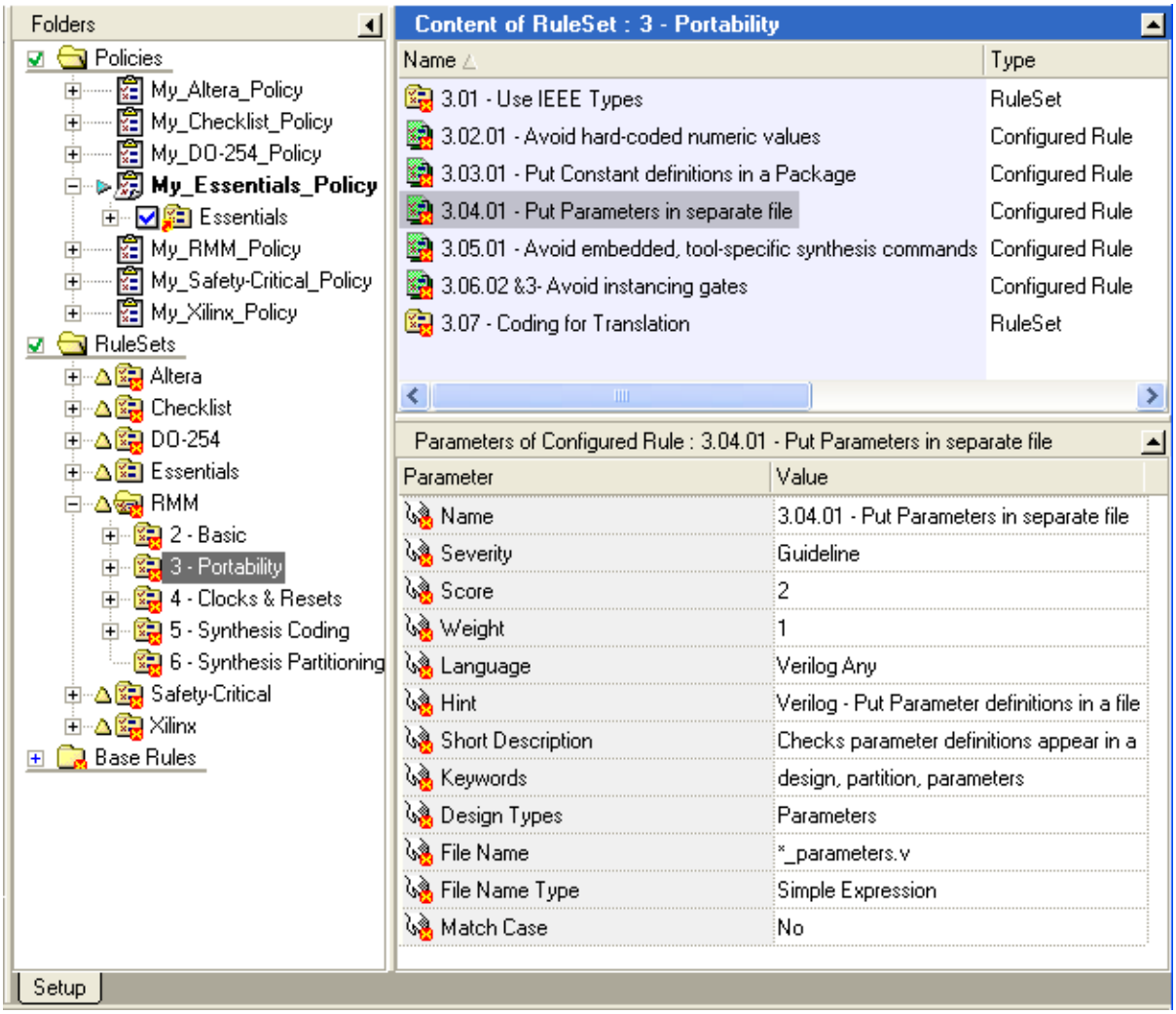
The *Checklist* ruleset provides the checks performed by 0-IN®; these checks can be applied to both VHDL and Verilog designs.

The *Safety-Critical* ruleset provides high-impact checks for safe coding practices to avoid typical design errors and to provide safe synthesis.

The *DO-254* ruleset extends the *Safety-Critical* ruleset to add code checks that ensure efficient code reviews.

Note that the rulesets provided with DesignChecker may differ according to the interface tool you are using.

The contents of ruleset folders can be viewed in a similar way to the *Base Rules* folder. For example, the following picture shows the content and parameters for the *Portability* folder in the *RMM* ruleset:



The Essentials Ruleset

By default, the *Essentials* ruleset runs on your code if you do not create your own rulesets or policies or choose other DesignChecker rulesets and policies. The *Essentials* ruleset provides a set of core rules that check your code for common coding practices, downstream tool problems, and reuse. These checks are common in the design community and have proven to be a very good indication of the quality of your code.

The *Essentials* rules are editable as you have the ability to make modifications in their content; that is to say, the *Essentials* rules are not read-only. You can directly apply your changes in the *Essentials* ruleset folder, instead of copying the rules to your own ruleset in order to change them.

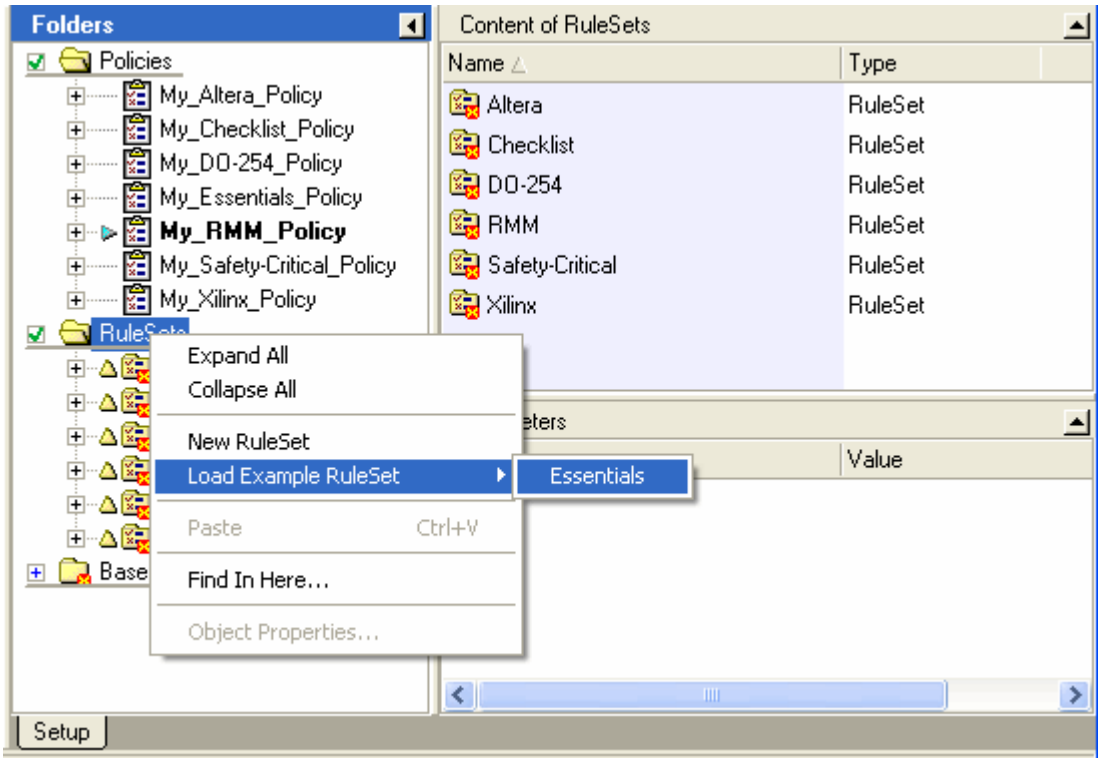
Note

Unlike the other built-in DesignChecker rulesets which are read-only (such as Altera, RMM, and Xilinx), the read-only overlay is not found on the *Essentials* ruleset folder indicating that it is configurable.

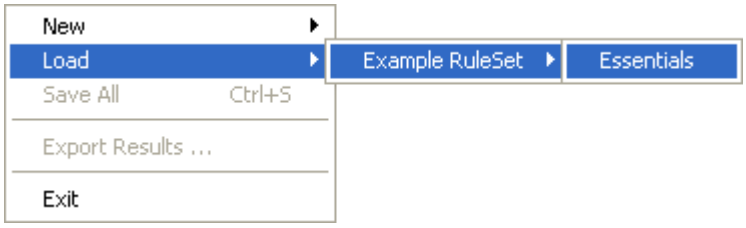
Restoring the Essentials Ruleset

If you modify or delete the *Essentials* ruleset, you can easily restore the original *Essentials* ruleset by doing the following:

1. In the Folders pane, right-click on the *RuleSets* folder and select **Essentials** from the **Load Example RuleSet** cascade.

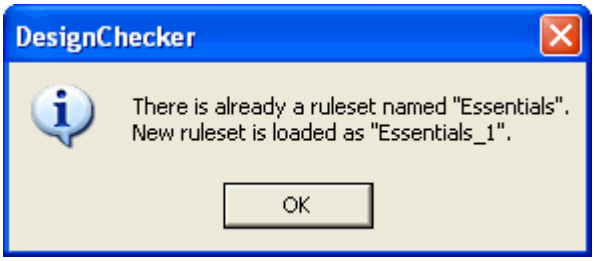


Alternatively, this step can be achieved from the **File** menu by selecting **Example RuleSet** from the **Load** cascade, and then selecting **Essentials** (note that the *Rulesets* folder must be selected first in the Folders pane, in order to have Essentials enabled in the File menu).

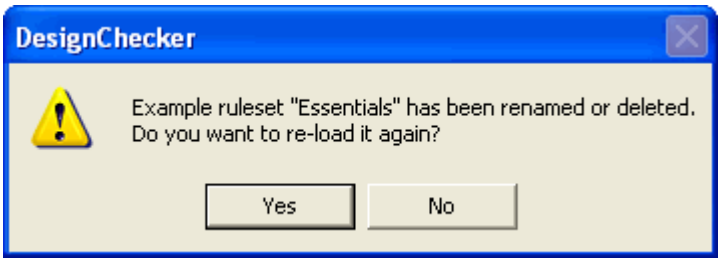


2. If the Essentials ruleset already exists with this name, it will be loaded with the same name suffixed by (_1); for example, the name of the loaded ruleset becomes *Essentials_1*. Note that the number following the underscore is incremental in case you re-load more than once.

   In that case, as shown in the following picture, a message is raised informing you of the existing ruleset's name as well as the name given to the newly loaded ruleset; click **OK**.



On the other hand, if the *Essentials* ruleset is renamed or deleted, a warning message is raised on re-invoking the DesignChecker tool to set up rulesets or policies. The warning message prompts you to confirm whether you wish to re-load the *Essentials* ruleset or not.



Reuse Methodology Manual (RMM) Ruleset

Reuse has become essential to achieving engineering quality and the timely completion of complex projects for modern electronic design. The *Reuse Methodology Manual* emerged from a collaborative project between Mentor Graphics and Synopsys, and has been widely adopted in the electronics industry as an effective methodology for creating reusable designs for System-on-a-Chip (SoC) design methodology.

A preconfigured *RMM* ruleset is supplied with DesignChecker, that you can use to check your code for reuse quality. The *RMM* is provided in a read-only format. You can also adapt the *RMM* ruleset to meet specific needs by creating a copy of the *RMM* ruleset and modifying it, in the same way that you configure the base rules.

Creating a Ruleset

You can create as many Rulesets as you need. You can create a new ruleset folder by using the  shortcut or by choosing **Ruleset** from the popup menu (or from the **New** cascade of the **File** menu).

If the *RuleSets* folder is selected in the Folders pane, a new ruleset is created at the top level. However, you can also create lower level rulesets by selecting any other folder. In the *RMM* ruleset for example, on expanding the *Portability* ruleset, you will find lower level rulesets for *Use IEEE Types* and *Coding for Translation*.

You can use the **Edit** or popup menu to **Cut**, **Copy**, **Paste** or **Delete** a ruleset or any of the base rules or configured rules used in a ruleset. You can rename a ruleset by directly editing its name or by choosing **Rename** from the popup menu.

Note

The same base rule can be copied any number of times to create as many configured rules as you require.

Note

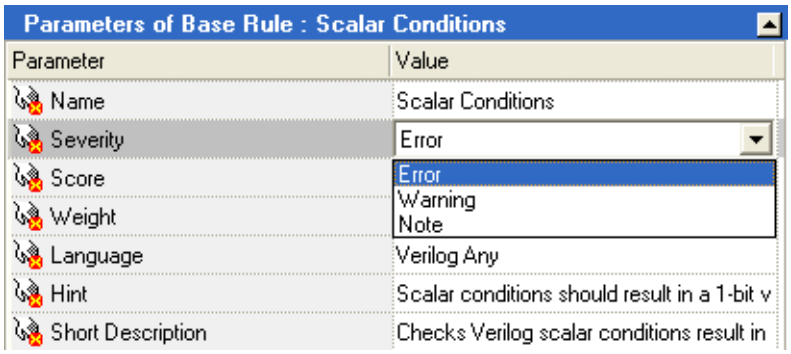
Modified rulesets and configured rules are shown with an overlay until they have been saved.

Configuring Rules

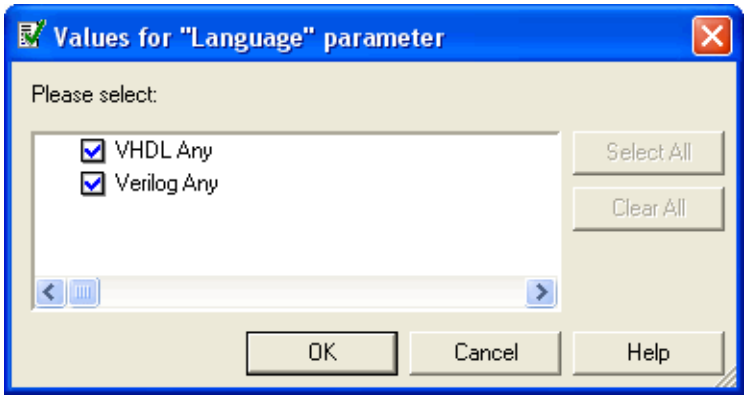
You can configure any of the rules in a ruleset by clicking on its default value in the Parameter pane.

DesignChecker has a variety of selection options. Some parameters can be configured by entering a text string or number. For example, to change the name of a rule, you can double-click on the default value of the Name parameter and then enter the new name; this also applies to other parameters such as the Short Description, Hint, etc. However, if you attempt to enter an invalid value or combination of parameter values, the rule is reset to its default values.

Some parameters provide a pulldown choice list. For example you can choose *Error*, *Warning* or *Note* for the *Severity* parameter.



Other parameters may display a list of options in a dialog box. For example, the following dialog box can be used to set values for the *Language* parameter:



The following parameters are common to all base rules:

- **Name** is an arbitrary, user-specified string which identifies the configured rule and must be unique within the ruleset.

  As mentioned earlier, you can rename a configured rule by double-clicking on the value of the Name parameter (so that it would be editable) and then typing the new name.

- **Severity** identifies the type of violation (Error, Warning or Note). The default is Error. You can create user-defined severitysets with different severity levels, and then assign the severitysets to their appropriate rulesets. For further information, refer to "Configuring Rule SeveritySets" on page 49.
- **Language** specifies the language (VHDL Any or Verilog Any).
- **Hint** provides information about the possible cause of the rule violation.
- **Short Description** is an optional user-specified string describing the configured rule.
- **Keywords** are terms indicative of the rule; they facilitate searching for the rule using the Advanced Search feature.

  You can add more keywords for the configured rule if required. By double-clicking on the Keywords row and opening the Parameters for Keywords dialog box, you can enter a string of keywords delimited by spaces in the Custom Parameter field. Alternatively, by clicking on the **Parameter File** button, a path can be specified to a file containing a list of keywords delimited by white space.

Note

Additional parameters are available for many rules. Consult the **Base Rule Reference Guide** available from the **Help** menu for the configuration options for each base rule.

Note

The Score and Weight parameters will also be available only if the option to perform design quality analysis is set. For further information about those two parameters, refer to "Design Quality Metric" on page 59.

You can access information about base rule parameters by choosing **Common Base Rule Parameters** from the **Help** menu to directly display the corresponding page of the *Base Rule Reference Guide* in your HTML browser.

You can directly access the description of the base rule corresponding to a configured rule by using the  button or by choosing **Base Rule Details** from the popup menu in the Content pane when the required rule is selected.

Setting Object Properties

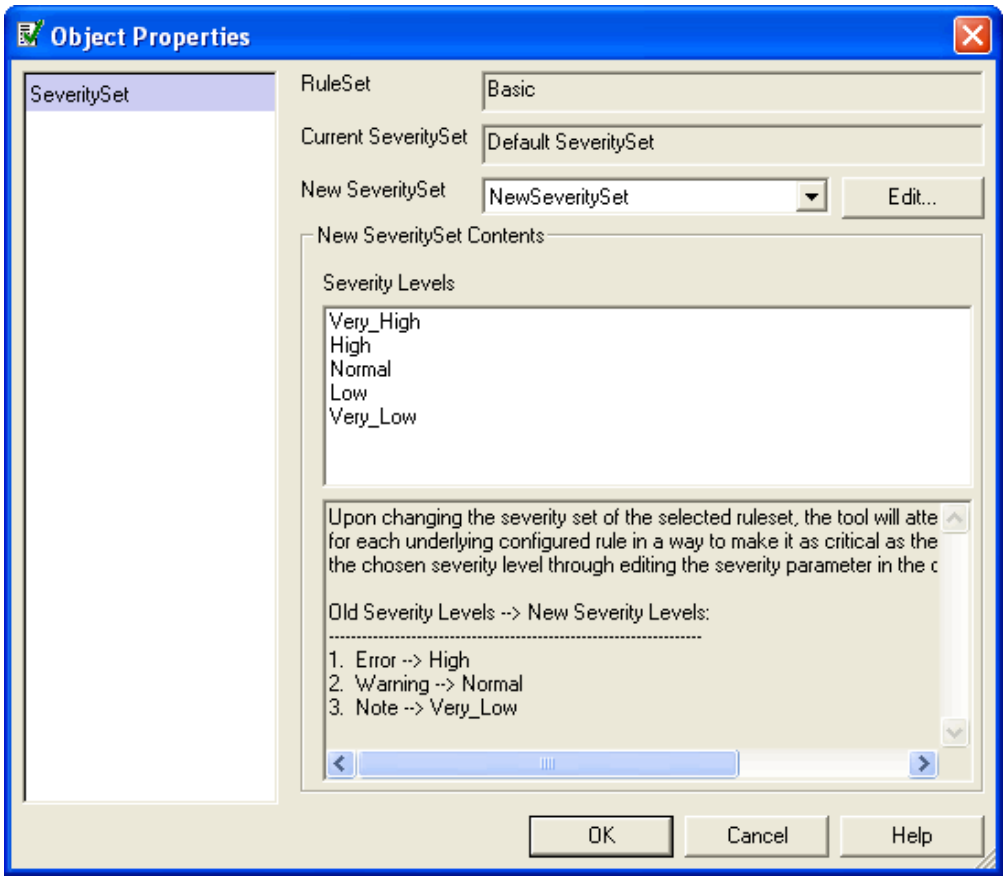
After creating the ruleset, you can set its properties through the Object Properties dialog.

Setting the SeveritySet Property

You can use the Object Properties dialog to assign the appropriate severityset to its ruleset. Refer to "Configuring Rule SeveritySets" on page 49 for further information on SeveritySets.

To set the SeveritySet property:

1. Open the Setup tab. In the Folders pane, open the *RuleSets* folder.
2. Right-click on a top-level ruleset and select **Object Properties**, or select **Object Properties** from the **Edit** menu. The Object Properties dialog opens.
3. Select “SeveritySet” from the list on the left hand side; consequently, the dialog shows the name of the ruleset and its current severityset. If you have not manually assigned a severityset before, you will find that the default severityset is automatically assigned to the ruleset.



4. From the New SeveritySet drop-down menu, select the severityset you want to associate to the ruleset. The drop-down menu contains all the previously user-defined severitysets in addition to the default built-in DesignChecker SeveritySet. Refer to“Creating a SeveritySet” on page 50 for information on how you can create user-defined severitysets.

   Note that after selecting the severityset, the following information is displayed by the DesignChecker:

   - The severity levels pertinent to the selected severityset are displayed in the white text area.

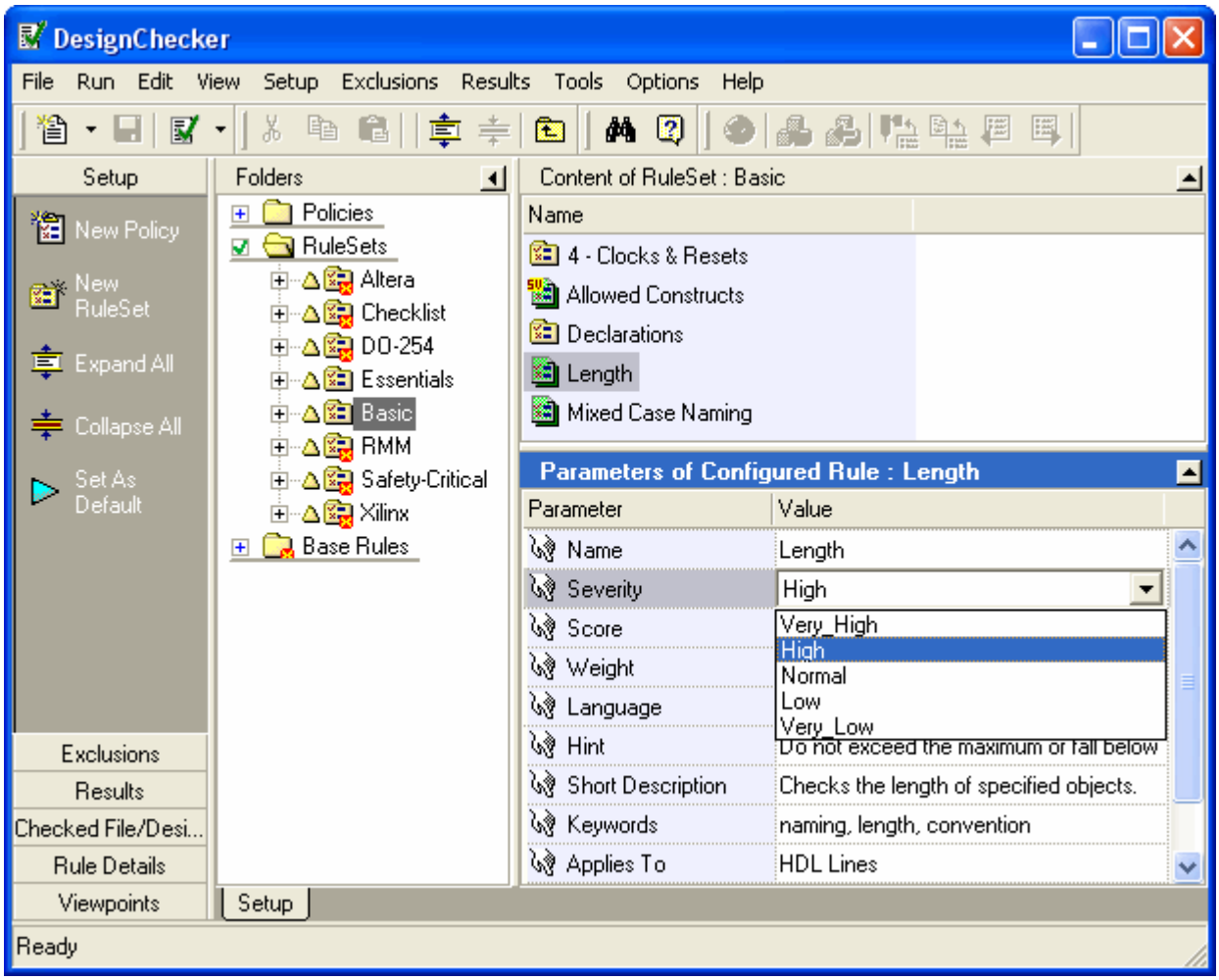
- The grey text area shows the mapping that shall be applied by DesignChecker between the old severity levels and the new ones.

5. Click **OK** and then save the ruleset.

Note

Modified rulesets and configured rules are shown with an overlay until they have been saved.

On selecting any configured rule from the ruleset, and checking its severity parameter, you will find the parameter values updated according to the severity levels of the assigned severityset.



It is important to observe the following remarks:

- Upon copying a base rule in the modified ruleset, the severity parameter of the rule is updated according to the severityset assigned to the parent ruleset.

- If a child ruleset is created for the modified ruleset, on opening the Object Properties dialog of the child ruleset, you will find that it has inherited the same severityset of the parent ruleset.
- Upon copying a configured rule from one parent RuleSet X to another parent RuleSet Y, the copied rule inherits the severity levels of the new parent ruleset Y. Furthermore, the severity value of the configured rule copied from RuleSet X is automatically mapped to a corresponding severity value in RuleSet Y.

  This is illustrated in the following example:

  a. RuleSet X is created and assigned the DesignChecker Default SeveritySet which constitutes of the three levels Error, Warning, and Note.

  b. In RuleSet X, three configured rules X1, X2 and X3 are created and given the severity values Error, Warning, and Note respectively.

  c. RuleSet Y is created and assigned a user-defined severityset that constitutes of five levels such as Very High, High, Normal, Low, and Very Low.

     Refer to"Creating a SeveritySet" on page 50 for information on how you can create user-defined severitysets.

  d. In RuleSet Y, the three configured rules X1, X2 and X3 are copied from RuleSet X.

     Hence, being under RuleSet Y, the three rules X1, X2 and X3 now inherit the severity levels Very High, High, Normal, Low, and Very Low. Furthermore, an automatic mapping occurs between the old levels and the new levels.

     The following table shows the mapping of severity values:

Configured Rule	RuleSet X	RuleSet Y
X1	Error	High
X2	Warning	Normal
X3	Note	Very Low

     However, on copying the three configured rules back to RuleSet X, the copied rules will be altered back to Error, Warning, and Note respectively.

Configuring Rule SeveritySets

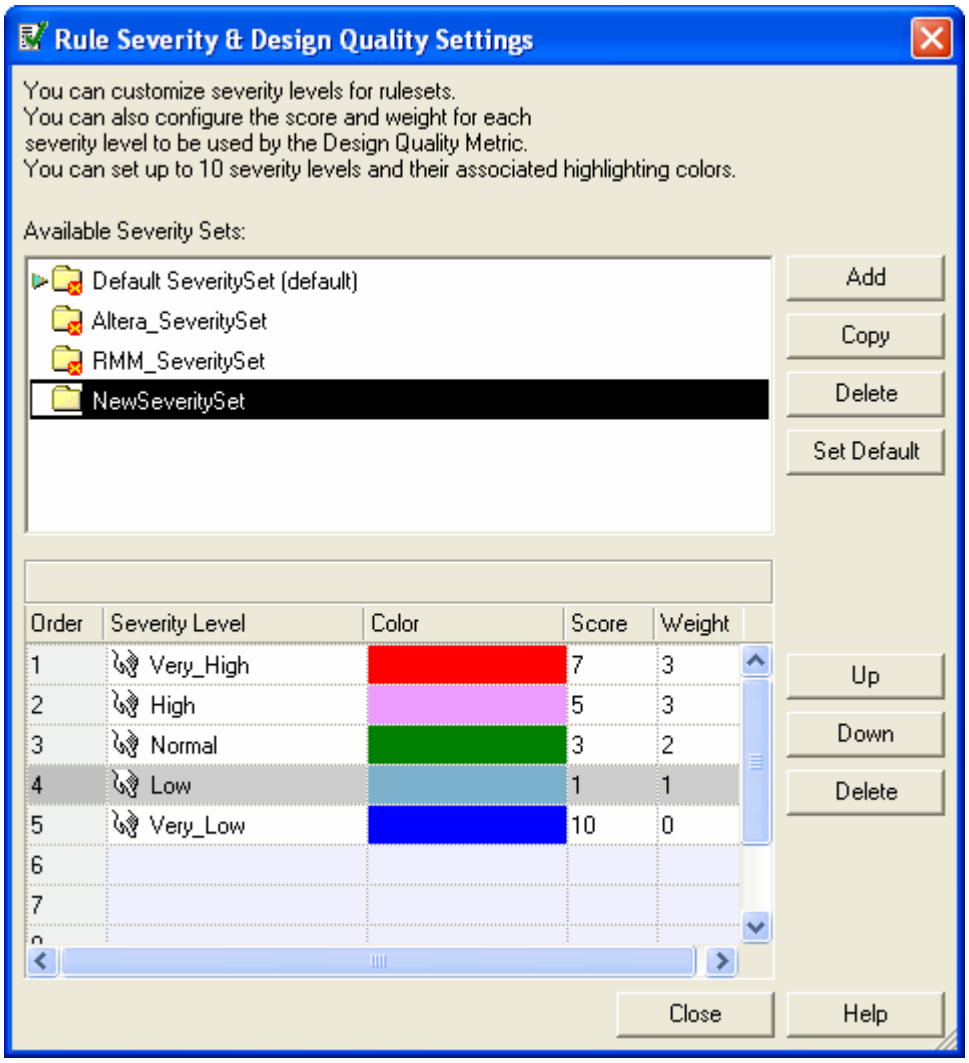
DesignChecker enables you to create user-defined severitysets. The SeveritySet is a group of severity levels that signify the criticality of the rule violation; each severity level is associated to a different color. The colors are used later in highlighting violations displayed in the Results tab after running the analysis. This feature helps you obtain more accurate results by identifying the exact measure of the rule violation severity.

Creating a SeveritySet

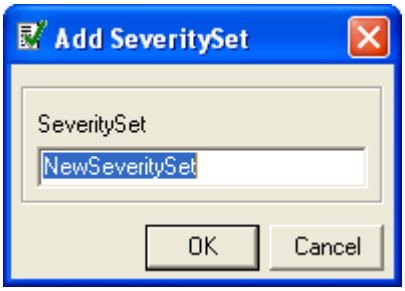
You can create as many severitysets as required to be applied to RuleSets. DesignChecker enables you to configure up to 10 levels for each severityset.

To create a SeveritySet:

1. Open the Setup tab. From the **Setup** menu, select **Rule Severity and Design Quality Settings**. The Rule Severity and Design Quality Settings dialog is opened. It constitutes of two main sections as follows:
   - The Available Severity list shows the previously user-defined severitysets, in addition to the default built-in DesignChecker SeveritySet. The DesignChecker SeveritySet is read-only.
   - The smart table shows the severity levels and colors of each severityset, in addition to the Score and Weight of each severity level (the score and weight are used only in the design quality analysis).



2. Click **Add** and use the Add SeveritySet dialog to enter the name of the new severityset. The new severityset is displayed in the Available SeveritySets list.



   Modifying the name of a severityset takes place through the Rule Severity & Design Quality Settings dialog. Select the severityset in the Available SeveritySets list, double-click on its name to show the cursor, and then type the new name.

3. From the Available SeveritySets section, select the newly added severityset; consequently, an empty smart table appears.

4. Use the smart table to enter the severity levels and their corresponding colors as follows:

   a. Double-click on the Severity Level cell, and enter the name of the severity level.

   b. Double-click on the Color cell, and select the corresponding color from the Select Color dialog.

   c. Double-Click on the Score and Weight cells to specify the score and weight of each severity level; they are used in the Design Quality Metric feature. For more information, refer to "Design Quality Metric" on page 59.

   d. To maintain the required order of levels, use the Up and Down buttons. It is recommended to arrange the severity levels from the most critical to the least critical. This ensures a proper mapping between severity levels on changing the severityset assigned to a ruleset.

   e. To remove a severity level, you can either click **Delete** or delete the name of the severity level directly from its cell.

Note

The table does not allow repeating the same severity level more than once.

5. To make the current severityset the default for rulesets, click the **Set Default** button adjacent to the Available SeveritySets list. By that, all the new RuleSets will automatically acquire the current severityset. The default severityset is indicated by a ▸ marker.

Note

By setting a severityset as the default, it is automatically assigned to any future created RuleSets only. However, this default severityset is not assigned to the already existing RuleSets unless you do that manually; refer to "Assigning a SeveritySet to a RuleSet" on page 53 for information on manual assignment.

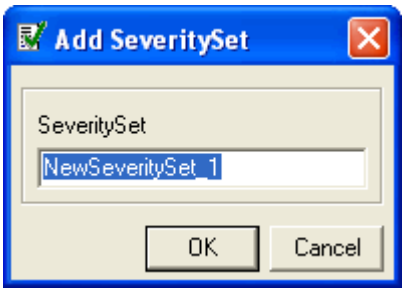
You can also configure quality scoring settings for severitysets through the Rule Severity and Design Quality Settings dialog box; refer to "Design Quality Metric" on page 59 for more information.

Copying a SeveritySet

DesignChecker enables you to make a duplicate of an existing severityset, thus, saving you the time of re-creating a similar severityset with the same levels and colors.

To copy a SeveritySet:

1. From the **Setup** menu, select **Rule Severity and Design Quality Settings**.
2. Select the severityset from the Available SeveritySets list.
3. Click **Copy** and use the Add SeveritySet dialog to enter the name of the new severityset. By default, the Add SeveritySet dialog displays the name of the original severityset suffixed by (_1); you can change this name if necessary.



The duplicate severityset is displayed in the Available SeveritySets section. On selecting the new severityset, the same severity levels and colors of the original severityset are displayed in the smart table.

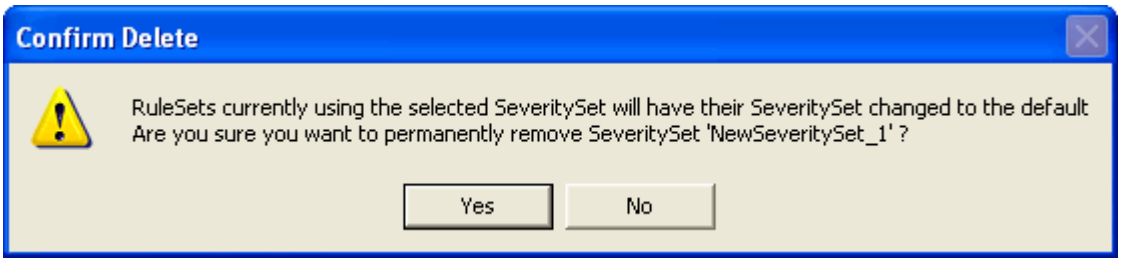
Deleting a SeveritySet

You can delete a previously defined severityset. If the deleted severityset is assigned to a ruleset, DesignChecker automatically changes the severityset of that ruleset to the default.

To delete a SeveritySet:

1. From the **Setup** menu, select **Rule Severity & Design Quality Settings**.
2. Select the severityset from the Available SeveritySets list.

3. Click **Delete**. A confirmation message is raised informing you that rulesets using the current severityset will have their severityset switched to the default; click **Yes** to delete.



Assigning a SeveritySet to a RuleSet

Having created severitysets, you can now assign the appropriate severityset to its ruleset. The severityset is a property of the ruleset that is set up through the Object Properties dialog. To open the dialog, right-click on the ruleset and select Object Properties. Refer to "Setting Object Properties" on page 46, for details on assigning severitysets.

Working with Policies

A policy specifies the rulesets that you want to run and allows you to enable or disable any of the configured rules or rulesets in the selected policy. You can display the default policy by expanding the *Policies* folder.

A referenced ruleset can be configured, while the ruleset it references remains unchanged. Consequently you can use referenced rulesets to build policies that reflect your development needs. The contents of any policy can be viewed in a similar way to the *Base Rules* and *RuleSets* folders.

Note

A referenced ruleset is indicated by the overlay.

Policies are basically created by team members. Yet, the project manager or team leader can create policies and share them among team members. The project manager can create policies, set them as read-only, and then copy the policies directory to a central server so that each individual in the team has access to these shared policies. Subsequently, the team members can reference the policies by setting the appropriate path. The steps of sharing policies differ according to the interface tool you are using. Refer to the integration documents available through the **Help** menu of the interface tool for further details.

Creating a Policy

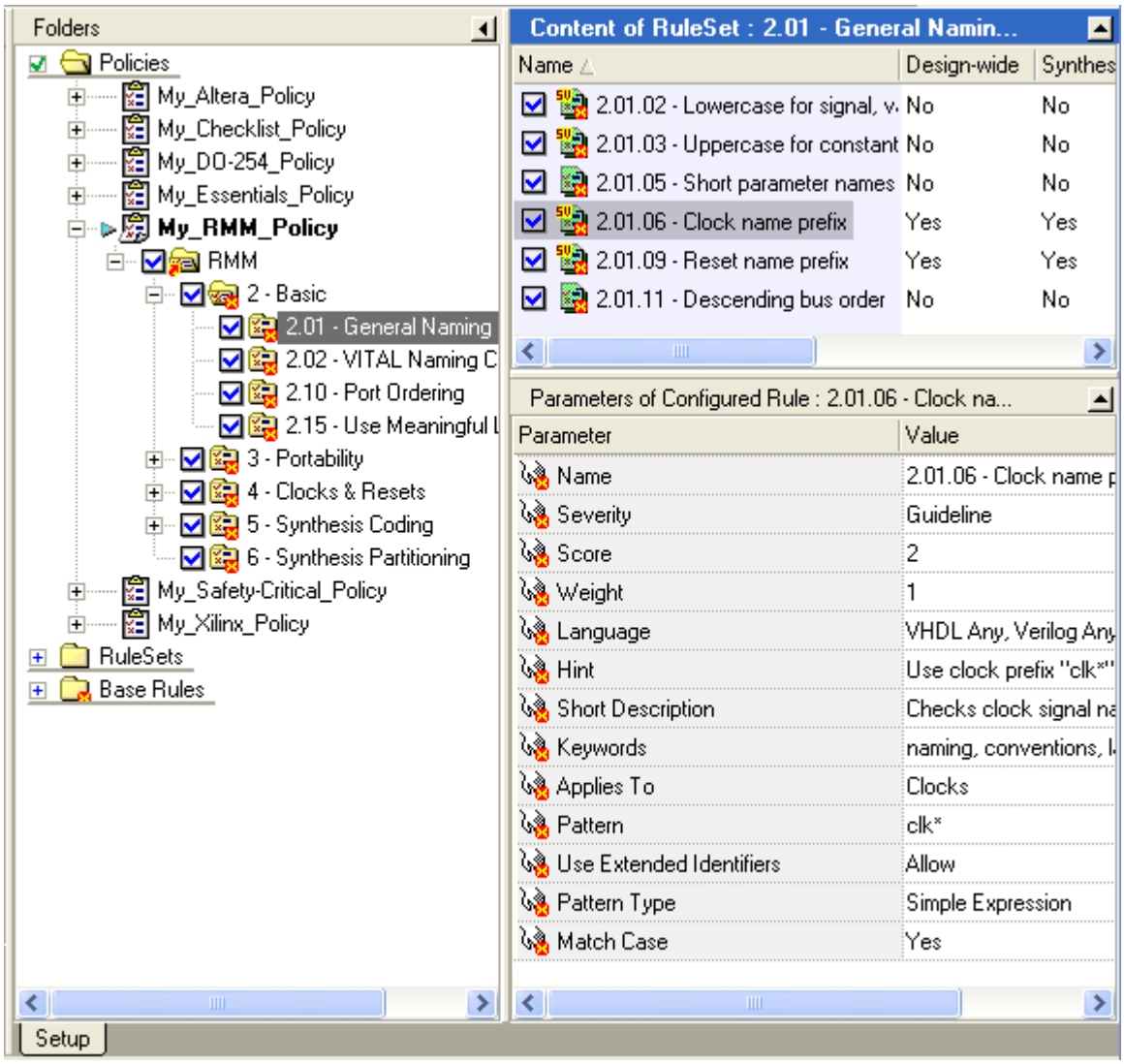
You can create a new policy folder by using the shortcut or by choosing **Policy** from the popup menu (or from the **New** cascade of the **File** menu) when the *Policies* folder is selected in the Folders pane.

You can use the **Edit** or popup menu to **Cut**, **Copy**, **Paste** or **Delete** a policy or any of the configured rulesets used in a policy. You can rename a ruleset or policy by directly editing its name or by choosing **Rename** from the popup menu.

You can also drag and drop a rule into a ruleset, or a ruleset into a policy.

You can enable or disable a rule or ruleset within a policy by setting the check box or by choosing **Enable** or **Disable** from the **Setup** or popup menu. If you unset one or more rules within a ruleset but there is still one or more enabled rule, the ruleset check box is shown with a gray check mark ☑. If the ruleset is fully enabled, the check box is shown with a blue check mark ☑. If a rule or ruleset is disabled, the check box is empty ☐. See “Adding Justification for Disabled Rules” on page 55.

The following picture shows the *Clock name prefix* rule in the *General Naming* ruleset:



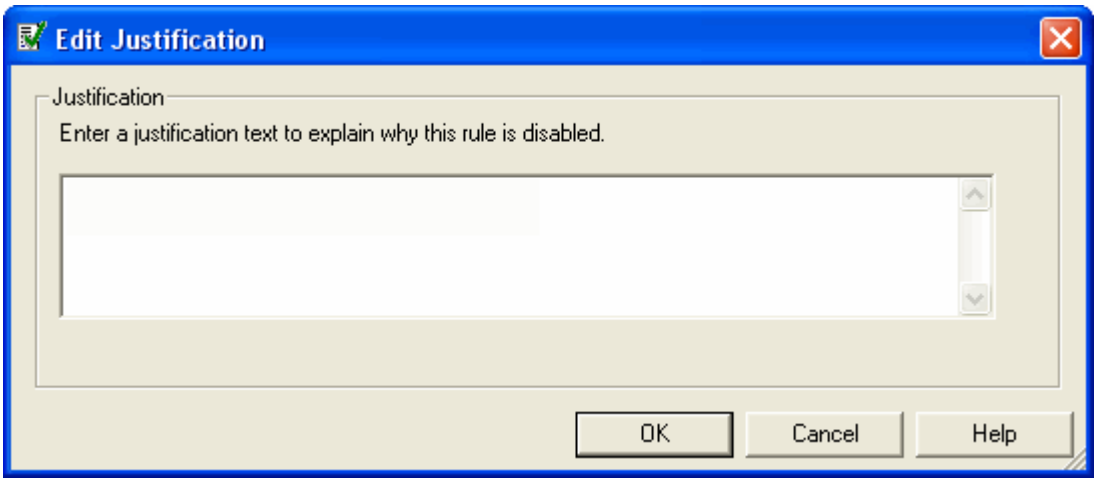
Note

Modified policies, rulesets and configured rules are shown with an ⓘ overlay until they have been saved.

Adding Justification for Disabled Rules

You can add a justification for disabling a ruleset/rule, for your own reference.

1. Open the policy you are using in the Setup tab.
2. Right-click on the disabled ruleset/rule in the content pane and select **Edit Justification** from the popup menu. A dialog box opens in which you can type a justification.



The justification will be displayed in the Rule Details tab after running the analysis (see The Rule Details Tab).

Setting the Default Policy

If you have created more than one policy, you can set the default policy by selecting the required policy and using the shortcut or by choosing **Set As Default** from the **Setup** or popup menu. Each time you run a DesignChecker analysis, the current default policy is applied.

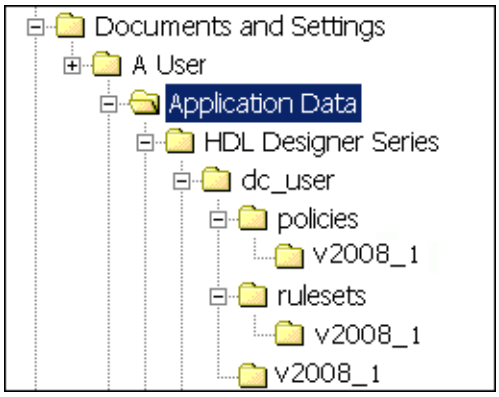
The default policy is indicated by a marker.

Saving Rulesets, Policies and Preferences

You can back out of changes to a policy or ruleset and replace it by the last saved version by choosing **Revert** from the popup menu.

You can save all modified rulesets and policies by using the button or by choosing **Save All** from the **File** menu. You can control whether a ruleset can be modified by setting the appropriate file permissions.

On Windows XP systems your rulesets and policies are saved by default in the DesignChecker directory within your user preferences. The following figure shows the location of the DesignChecker directory with HDL Designer Series for example.



On Unix and Linux systems, your rulesets, policies and preferences are written within *$HOME/<interface tool name>/dc_user*.

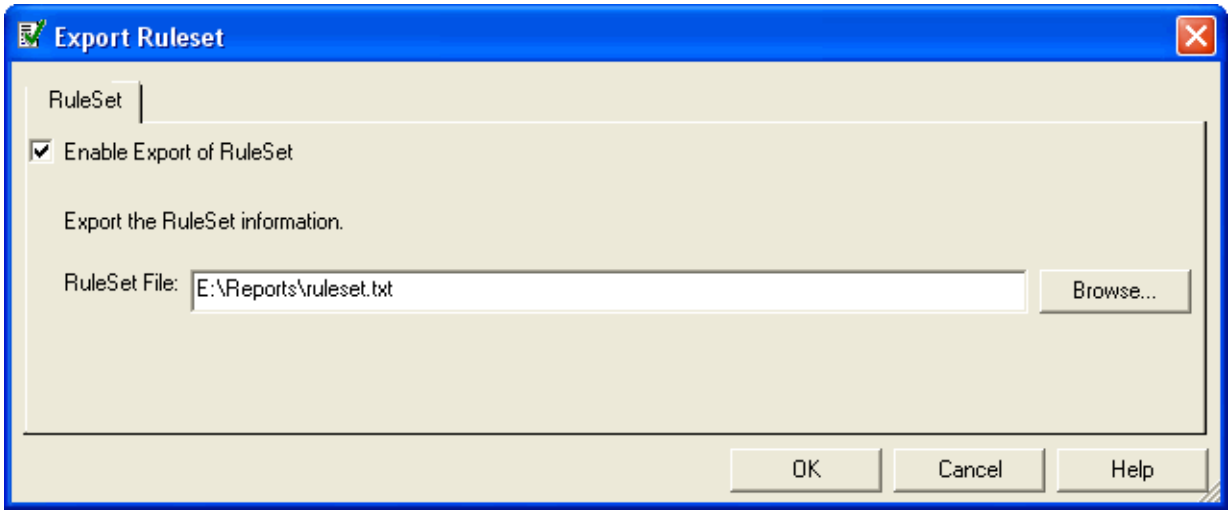
DesignChecker preferences include information about your settings and the last viewpoint you selected.

Note

You must have write permissions to your preferences location to save a modified ruleset.

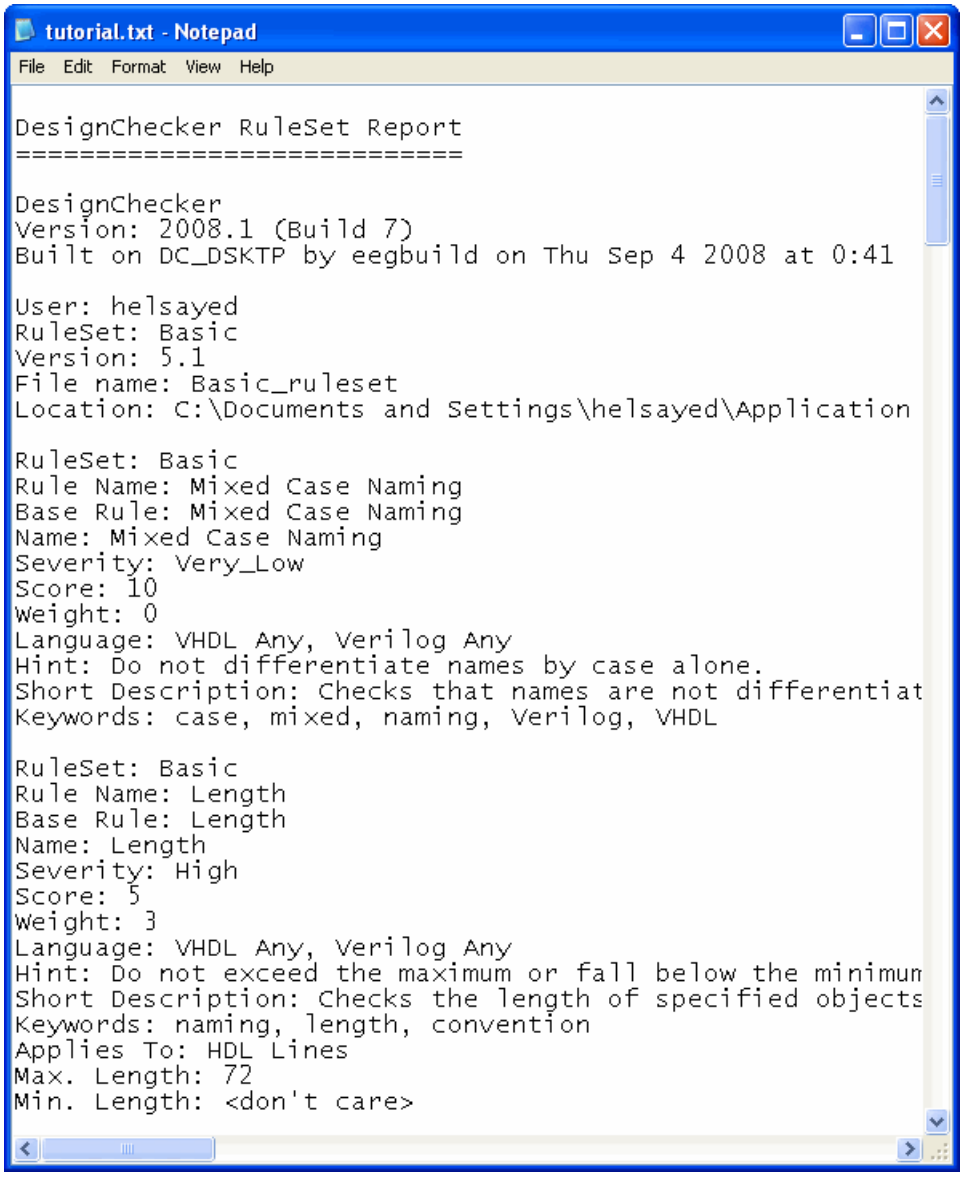
Exporting Rulesets

You can export a copy of your Ruleset as a file using the **RuleSet Report** command from the **Setup** menu. This facility is only available from the Setup tab, when a ruleset has been selected. You must enable the export and specify a destination path and filename in the Export Ruleset dialog box.



The report can be generated in *.txt* format.

The exported text file contains full details of your ruleset. For example, an exported ruleset report is shown below.



Design Quality Metric

The Design Quality Metric feature enables you to measure the quality and reusability of your design. This feature allows you to set up quality scoring standards in DesignChecker in order to identify how far your code adheres to your company's quality standards, and thus measure the effectiveness of your code if reused.

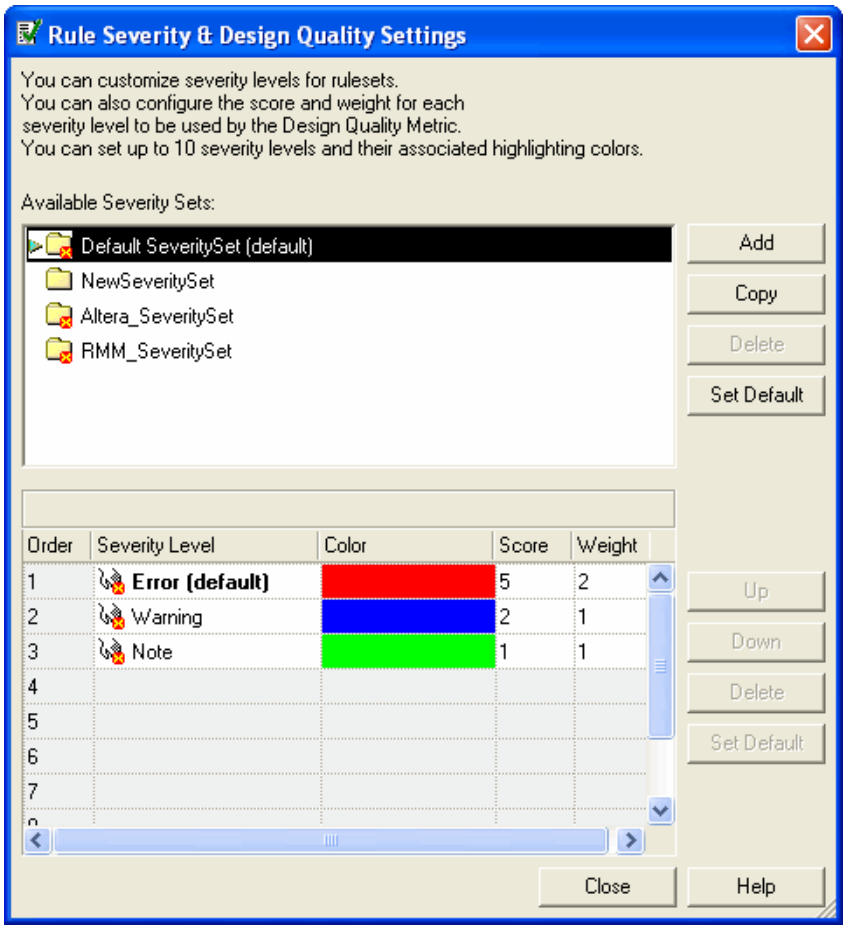
This is achieved in DesignChecker through three main phases: configuring your quality scoring standards through setting up a user-defined severityset, calculating quality using the scoring standards of the active policy, and reporting the quality scoring.

Configuring Quality Scoring Settings

Setting up your quality scoring standards takes place through defining a SeveritySet in which you define a score and weight for each severity level in the set. These scores and weights are used to calculate the final quality score of your design. For further information about user-defined severitysets, refer to "Configuring Rule SeveritySets" on page 49.

To set up quality scoring:

1. From the **Setup** menu, select **Rule Severity and Design Quality Settings**.
2. In the Rule Severity and Design Quality Settings dialog box, select the required severityset from the Available SeveritySets list. You can define a new severityset if necessary and set its severity levels; refer to "Creating a SeveritySet" on page 50 for more details.



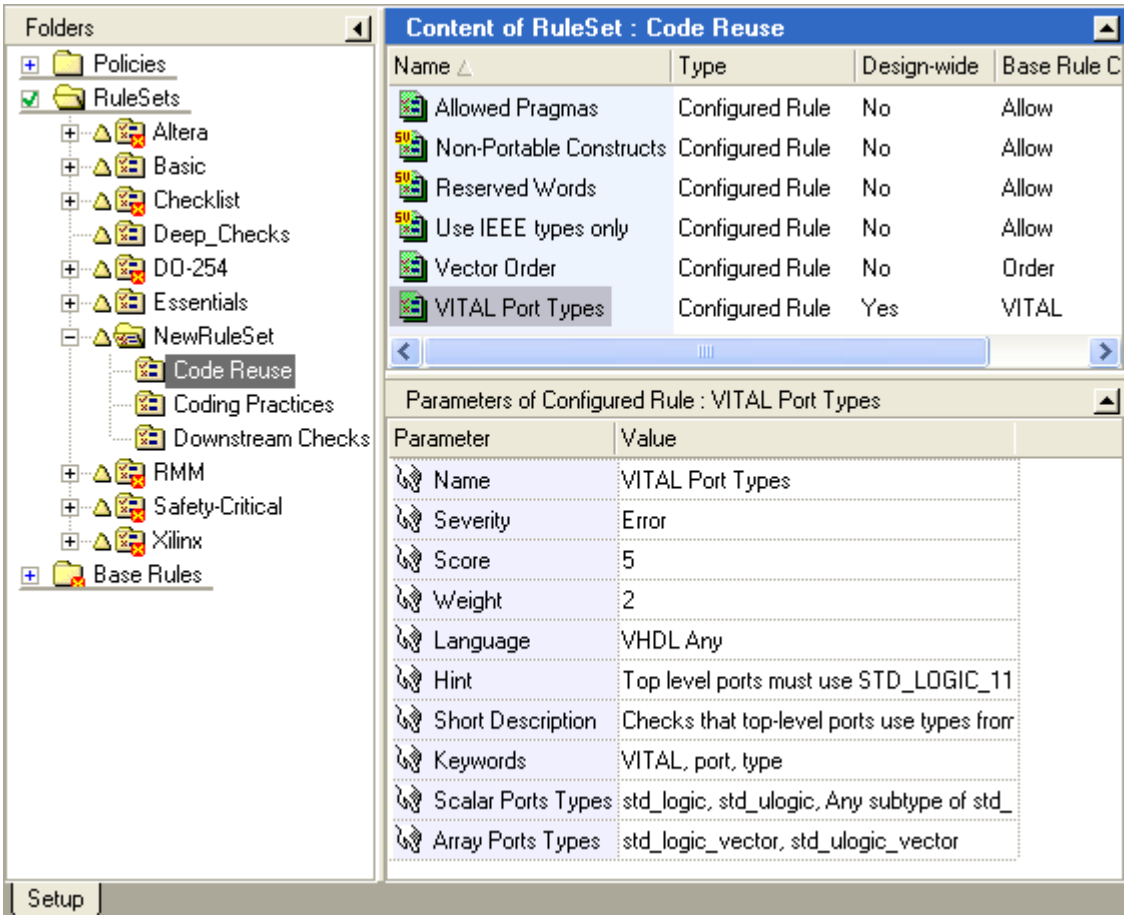
3. In the smart table, define a Score and Weight for each severity level in the respective columns.

Note

The preconfigured DesignChecker severitysets - namely the Default, Altera, and RMM SeveritySets - have a default score and weight which cannot be modified.

On assigning a severityset to a ruleset, all the children rules are by default given the same score and weight of the rule's severity level. If you display the parameters of the rule, you will find the score and weight corresponding to the rule's severity as shown in the following picture.

Note that if scoring is disabled in the Design Quality Options dialog box, then the Score and Weight parameters will not be displayed in the Parameters pane.



You can directly modify the score and weight parameters for a specific rule to override those of the severityset. On the other hand, if you modify the severity parameter of the rule, the score and weight parameters will be updated accordingly.

Also, if you make global modifications to the score and weight of the SeveritySet itself in the Rule Severity and Design Quality Settings dialog box, this will in turn affect the score and weight of any previously associated rules.

Note

It is important to note that when you define a severityset in the Rule Severity and Design Quality Settings dialog box and set it as the default, only newly created RuleSets will be automatically assigned this default severityset and thus inherit its quality scoring settings. In case of any RuleSets that have been created prior to setting this default severityset, you will have to manually assign the severityset in order for these RuleSets to inherit the new global severity and scoring settings. Refer to "Creating a SeveritySet" on page 50 and "Assigning a SeveritySet to a RuleSet" on page 53 for information.

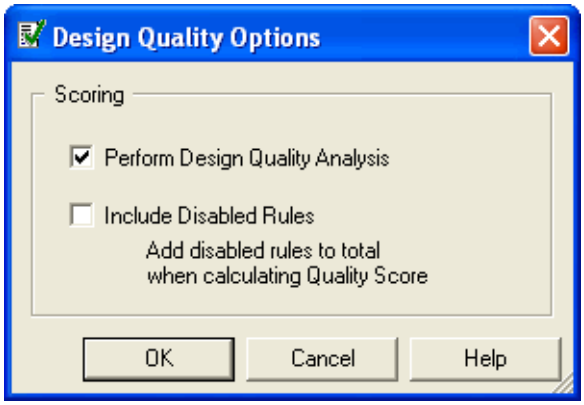
Calculating Quality Scoring

Quality calculation occurs on running an analysis for your design. DesignChecker estimates the Quality Score percentage of your design using the total possible score and the total actual score. The total possible score is the sum of the possible scores (score * weight) of all the rules in the policy. On running the analysis, if any rule fails, then it takes a score of zero; the total actual score is hence calculated by subtracting the product of the score and weight values (possible score) for each failing rule from the total possible score. The final Quality Score is the quotient of the total actual score and the total possible score as a percentage value that is rounded to the nearest tenth.

For example, Rule X has a score of 5 and a weight of 2, then the possible score for Rule X is 10; Rule Y has a score of 3 and a weight of 1, then the possible score for Rule Y is 3; the total possible score is therefore 13. On running analysis, Rule Y was violated and therefore its possible score was subtracted to give a total actual score of 10. Based on the total possible score of 13 and the total actual score of 10, the final Quality Score is 77%.

To calculate the design's quality score, do the following:

1. Enable the Design Quality Metric feature by doing the following:
   a. From the **Options** menu, select **Design Quality Options**.
   b. In the Design Quality Options dialog box, select the option Perform Design Quality Analysis.



You can choose to “Include Disabled Rules” in order to add the possible score (score * weight) of all available disabled rules to the total possible score during calculation. However, disabled rules are given an actual score of zero.

In this case, the final Quality Score is also the quotient of the total actual score and the total possible score (which now includes the possible scores of disabled rules) as a percentage value that is rounded to the nearest tenth. If this option is not selected, then the possible score of the disabled rules will be excluded from the total possible score.

Following the example mentioned above, if there is Rule Z which has a score of 2 and a weight of 1, then its possible score is 2. If Rule Z is disabled and you have chosen to Include Disabled Rules, then the total possible score becomes 15 and the total actual score remains as 10. The final Quality Score is thus 67%.

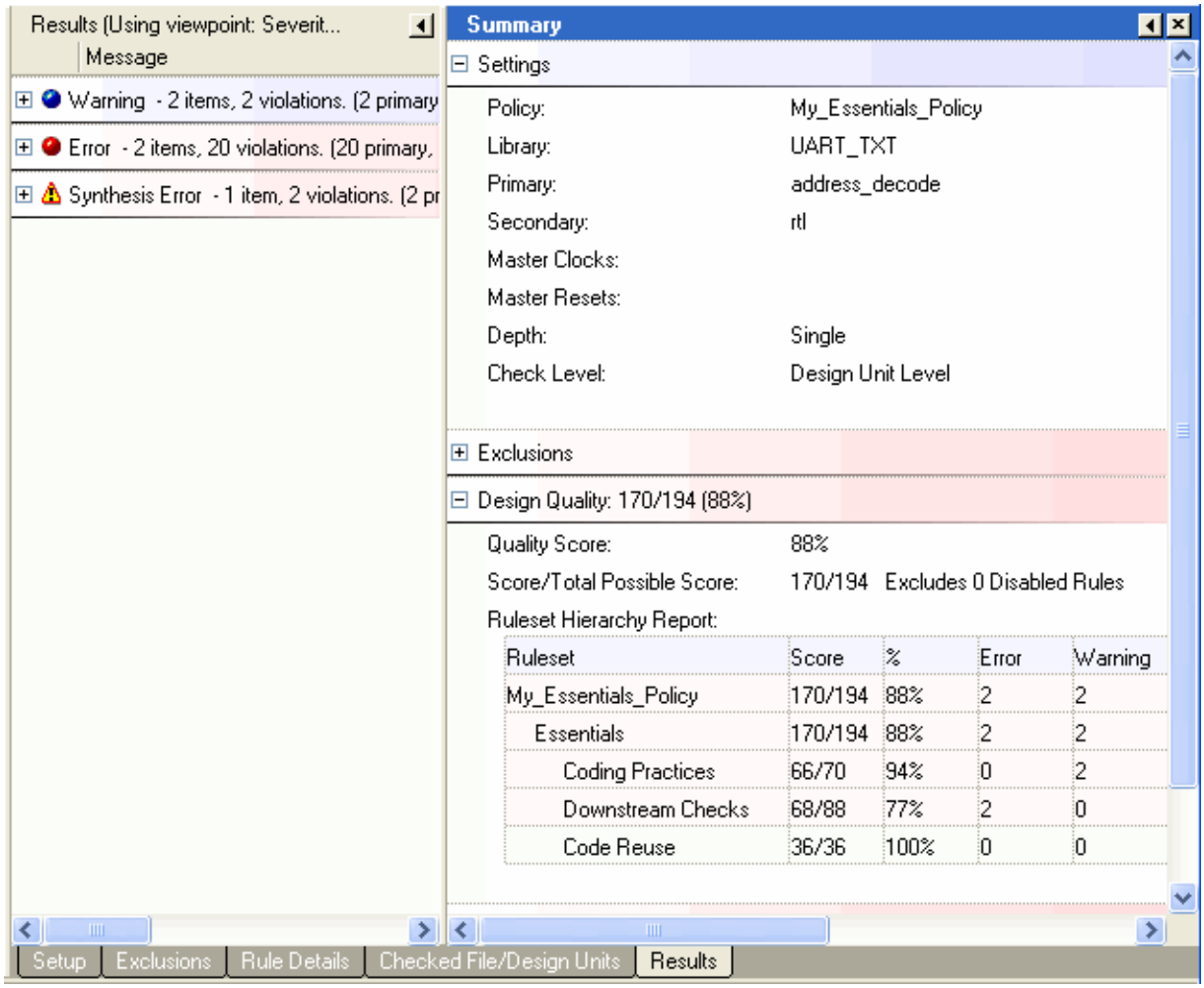
2. Make sure that the rulesets in the current active policy are assigned to the appropriate severitysets (with the scores and weights set as required; refer to “Configuring Quality Scoring Settings” on page 60).

Note

As the score and weight parameters of an individual configured rule can override the global score and weight preset for the SeveritySet, DesignChecker hence performs calculation based on those of the configured rules in the ruleset.

3. Run the analysis after having selected the design items in the interface tool first.

The Results tab opens displaying the outcome of the analysis; the quality scoring is reported in the Design Quality section of the Summary pane.



For information about obtaining the results of the design quality analysis, refer to "Reporting Quality Scoring" on page 87 and "Exporting Quality Reports" on page 98.

Setting Exclusions

The Exclusions feature enables you to exclude certain areas of your code from being analyzed or to exclude certain rules or rulesets from being run whether on the entire design or on specific design items or files within your code.

Skipping certain areas of your code from being generally checked, or more specifically from being checked by certain rules, saves you the hassle of having to go through the violations that may be generated by this code, which might be in fact behaving as designed or is third-party code which does not comply to your coding standards.

DesignChecker has the following types of exclusions:

Table 2-2. Exclusion Types

Exclusion Type	Description
Code/Rule Exclusion	Excludes specific checks from being performed on specific parts of the design.
Pragma Exclusion	Allows you to skip specific code blocks.
Black Box Exclusion	Ensures that DesignChecker recognizes that a component is present, but does not apply any checks to it.
Don’t Touch Exclusion	Ensures that a file is not loaded or analyzed by DesignChecker.
File/Folder Exclusion	Excludes a specific file or folder from being analyzed.

Note that the exclusion types offered by DesignChecker differ according to the interface tool. For example, in case of using DesignChecker with Certe Testbench Studio, you will be able to apply File/Folder Exclusions to your Certe projects; in case of using DesignChecker with HDL Designer Series, you will be able to apply Code/Rule Exclusion, Pragma Exclusion, Black Box Exclusion, or Don’t Touch Exclusion.

Once you run an analysis, an Exclusions tab is displayed allowing you to review all the exclusion settings that have been applied to your analysis.

For further information on exclusions and how they can be configured, refer to the integration documents available through the **Help** menu of the interface tool.

Chapter 3
Running DesignChecker and Working with Results

This chapter describes how to run DesignChecker and how to use the different reporting options provided in the tool. It includes general procedures for using the DesignChecker Results tab, the range of results available, and the ways in which these results can be manipulated and exported in different formats. Additionally, it provides information on the Exclusions, Rule Details and Checked Files/Design Units tabs.

Selecting Files/Design Items for Analysis

You can specify the design or part of the design you wish to analyze through the interface tool you are using. You can also run the DesignChecker analysis through the interface tool. Refer to the integration documents available through the **Help** menu of the interface tool.

Once you run the analysis, the following tabs appear in DesignChecker: The Results Tab, The Exclusions Tab, The Rule Details Tab, and The Checked Files/Design Units Tab.

Note

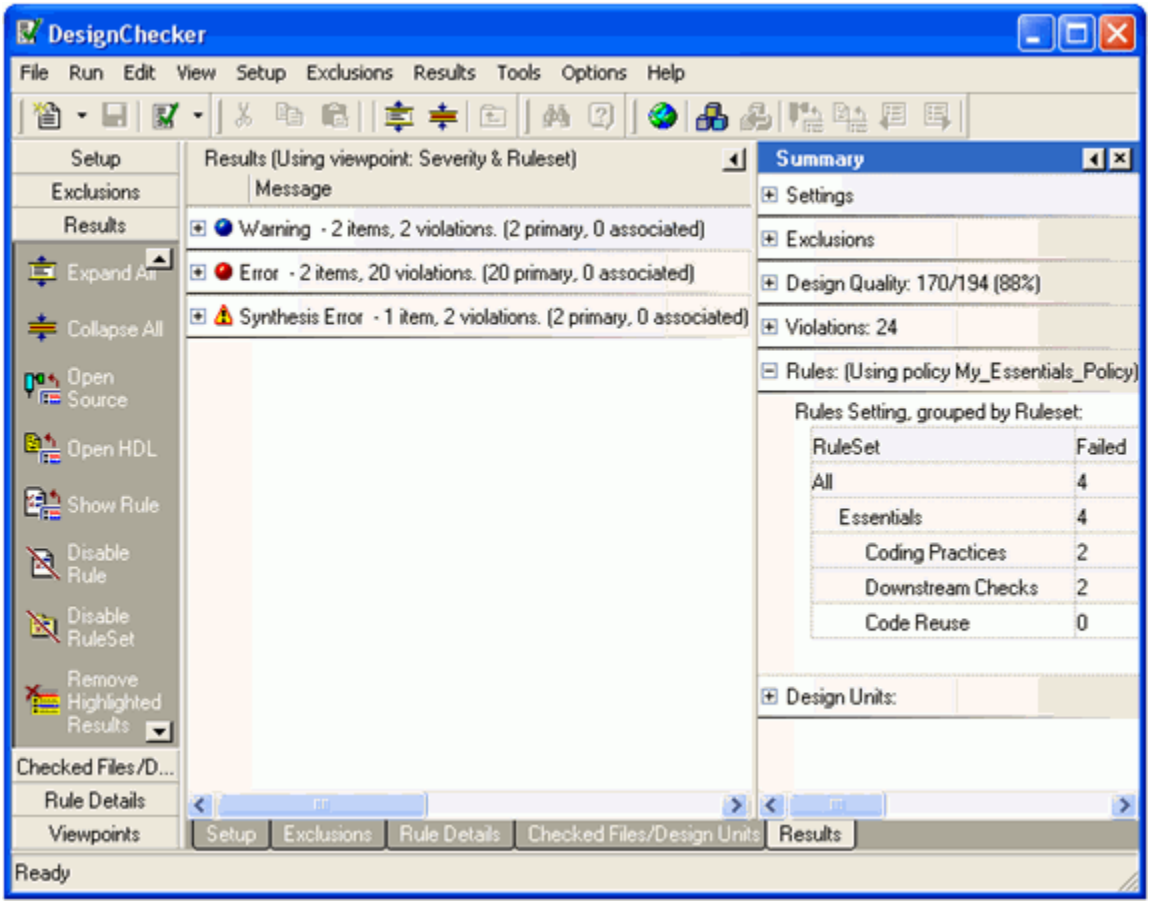
Before running DesignChecker, ensure that you have set the policy you wish to apply as your default policy.

The Results Tab

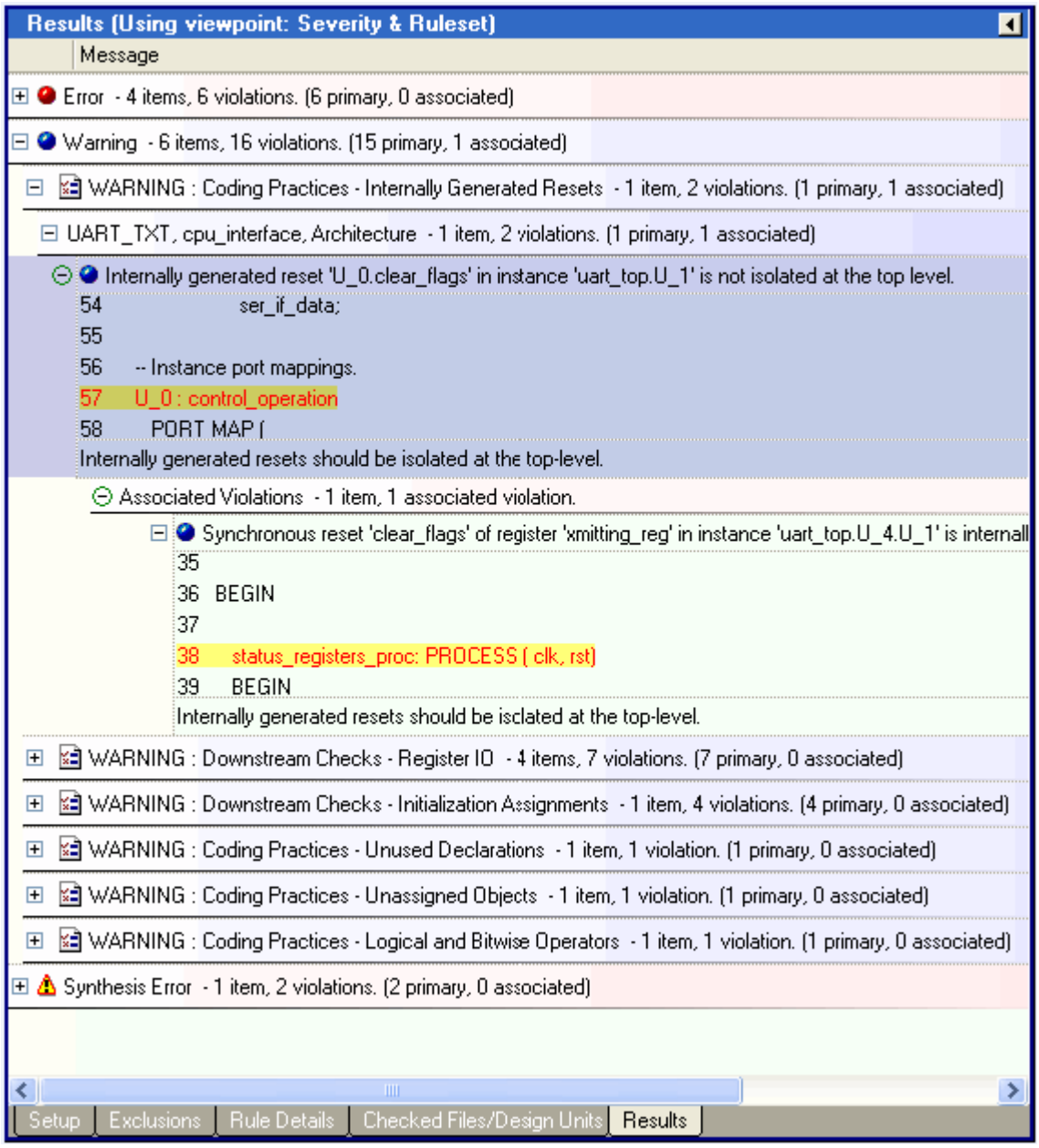
Once you have run DesignChecker, the results tab is displayed. This contains facilities to group, sort, or filter your results and to cross-reference error messages to the source files.

The Results tab is displayed automatically when you run an analysis, you can return to it at any time by selecting the Results tab in DesignChecker. The Results tab is divided into a results viewpoint and summary pane, as shown in the picture below. The summary pane (shown on the right) contains high-level information about the files analyzed, rulesets and policies applied, and the violations identified. Refer to "The Results Summary Pane" on page 85 for further information.

The viewpoint (shown on the left) displays the detailed results in a smart table. The content and appearance of the table are based on the settings of the current default viewpoint; refer to "Using Viewpoints" on page 74 for further details. For example, the following picture shows the results displayed in the *Severity & Ruleset* viewpoint after an analysis has been performed. You can use this pane to explore and examine specific occurrences of code violations by clicking on the ⊞ icon. DesignChecker also has the capability to automatically cross-reference and open the source file of any identified item.



As illustrated in the following picture, there are two types of violations in DesignChecker: primary and associated. The primary violation is the main violation; it might be a single independent violation, or it might involve other associated violations. Therefore, in that sense, associated violations are secondary violations grouped under one primary violation; they represent further details relevant to the primary violation.



Associated violations can represent the children violations that resulted from the primary violation, they can represent the instances in which the primary violation occurred, or they can represent parts of a compound primary violation, etc.

Results Tab Notation

The following icons are used to identify objects in the results tab:

Table 3-1. Results Tab Notation

Icon	Description
	VHDL architecture
	VHDL entity
	Verilog module
	File
	VHDL configuration declaration
	VHDL package body
	VHDL package header
	Error message (red)
	Warning message (blue)
	Note message (green)

Information is displayed using multi-column, multi-row “smart-tables” which support multiple alternative viewpoints for column selection, grouping, sorting and filtering.

You can change the content and format of the results display using viewpoints as described in “Using Viewpoints” on page 74.

You can display information about each result in a popup object tip by moving the cursor over the result rows.

Note

You can enable or disable object tips by setting the **Object Tips** option in the **View** menu.

Using the Results Tab

You can use the Results tab as the starting point for examining, reviewing and updating your designs. The powerful cross-referencing capabilities of DesignChecker enable you to navigate directly to areas of your design which have been highlighted for attention. You will see your source code opened and the relevant area highlighted in the editor. With some interface tools, you can also open the relevant area in an appropriate graphical editor.

Expanding and Collapsing Results

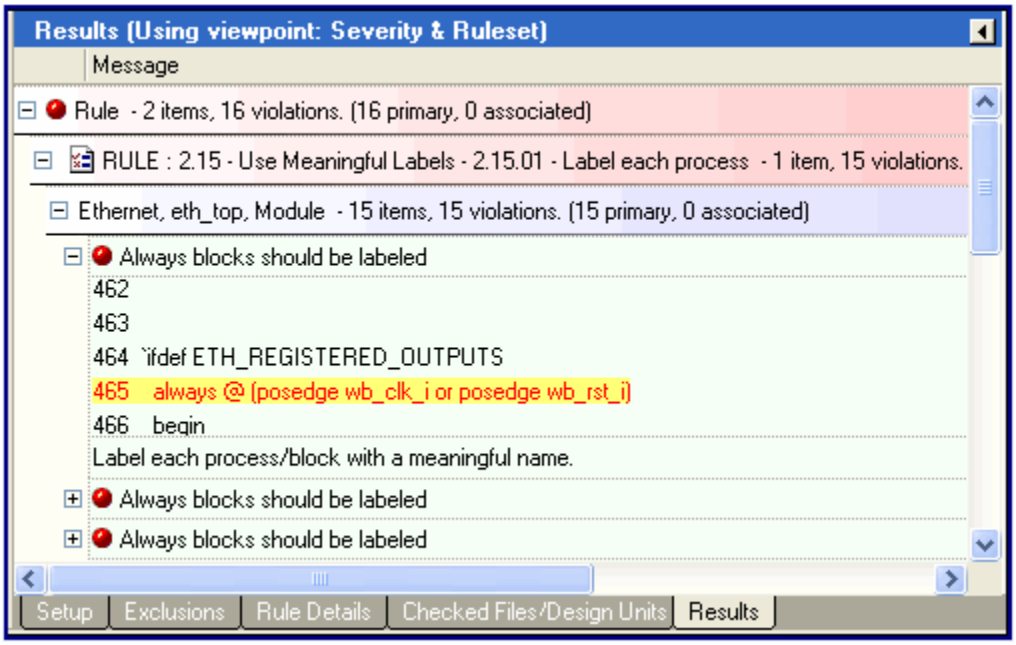
In the Results tab, when the default viewpoint is active, you can use the ⊟ or ⊞ icons buttons to drill down from the high-level summaries of results to specific issues. You can do this by expanding each result row and displaying the suggested correction and the code fragment where a rule violation occurs.

You can expand or collapse the selected rows by choosing **Expand From Here** or **Collapse From Here** from the popup menu.

You can expand all rows by using the  button or by choosing **Expand All** from the **Edit** or popup menu.

You can collapse all rows by using the  button or by choosing **Collapse All** from the **Edit** or popup menu.

The following picture shows expanded results for the *eth_top* example design (from HDL Designer Series) displayed using the default *Severity & Ruleset* viewpoint. The violation itself is highlighted and shown in its context. You can expand the contextualized view of the code snippet within the results table by dragging the upper or lower boundaries of the results cell with your mouse.



Note

You can open any section of code for direct editing in the corresponding text editor by double-clicking in the Results tab cell.

You can move to the next rule violation message (automatically expanding the results row if necessary) by using the  button or by choosing **Next Message** from the **Results** menu.

You can move to the previous rule violation message (automatically expanding the result row if necessary) by using the  button or by choosing **Previous Message** from the **Results** menu.

Copying Results

DesignChecker enables you to copy a violation message to the clipboard so you can paste it in any external editor. To do that, select a violation, and then right-click and choose **Copy Message** from the popup menu. Multi-selection of violations is allowed by holding down the **Ctrl** or **Shift** keys.

Note

If the violated rule belongs to the Report base rules, then you will copy the message and the report as well.

Opening Design Views from the Results Tab

DesignChecker has powerful cross-referencing capabilities. You can open the source file corresponding to a result row by choosing **Open HDL**/**Open Source** from the popup menu. For further information, refer to the integration documents available through the **Help** menu of the interface tool.

Showing the Setup Tab

You can open the setup tab showing the configured rule, ruleset and policy that corresponds to the selected result by choosing **Show Rule** from the **Results** or popup menu.

Note

You can access the description of the corresponding base rule by using the  button or by choosing **Base Rule Details** from the popup menu.

Viewing Severity Levels

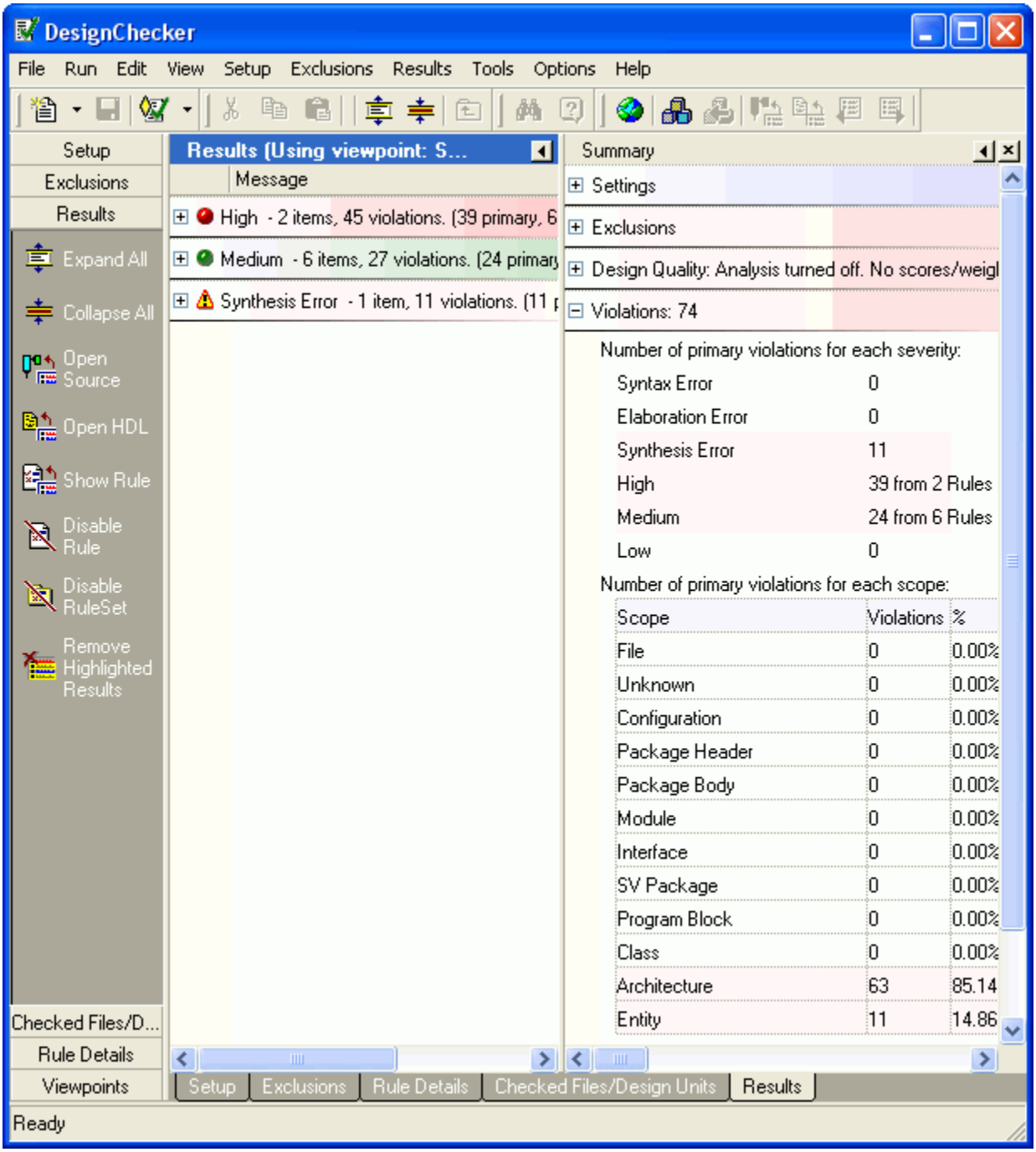
If there are severitysets assigned to rulesets, they are reflected on the Results tab after running the analysis. According to the configuration of the severityset, the severity levels and colors are used in highlighting violations displayed in the Results tab. Refer to "Configuring Rule SeveritySets" on page 49 for information on severitysets.

The severity levels affect both the Results tab and the Results Summary pane as follows:

- In the Results tab, the Severity column shows the severity levels of the rules for which violations have been found (for example: high, medium, etc.). Note that the background color of the violation reflects the color predefined in the severityset.
- In the Results Summary pane, the number of violations of each user-defined severity level is displayed.

To view Severity Levels:

1. In the Setup tab, create a new policy and add to it the ruleset to which the user-defined severityset is assigned. Set the created policy as the default policy. Refer to Working with Policies "Working with Policies" on page 54 for further information.
2. Run the analysis. The Results tab opens displaying the violations highlighted according to the severity levels of the configured rules and their corresponding colors.



Note that through the Summary pane, you can view the number of violations per severity level in the Violations section. Furthermore, on double-clicking on a specific severity level in the Violations section, you will be able to view the violation details pertinent only to that severity level in the Results pane.

Note

If the default policy used in running the analysis includes rulesets that are associated to different severitysets, then all the severity levels of those severitysets are shown collectively in the Violations section of the Summary pane.

Understanding the Types of Violations

In the Results tab, each violation is given a certain severity level depending on the prior configurations of your ruleset. For example, if the ruleset is using the severity set titled *Default SeveritySet*, the violations will have the severity levels *Error*, *Note* or *Warning*, and if the ruleset is using the severity set titled *RMM_SeveritySet*, the violations will have the severity levels *Rule* or *Guideline*. Refer to "Configuring Rule SeveritySets" on page 49 for further information.

However, apart from the current severity set used in your analysis, you may find some violations having different titles such as *Syntax Error*, *Synthesis Error* or *Elaboration Error*. For more information on these errors, refer to Design Correctness and Synthesizability.

Controlling the Display of Results

This section describes how you can manage the layout and organization of the displayed DesignChecker results.

Using Viewpoints

A default viewpoint is used in the results tab to display the messages issued by DesignChecker. Viewpoints are used to control how the results are displayed and persist that information between work sessions.

The last viewpoint used is stored in your preferences and automatically re-opened when you invoke a new session so that the same columns are displayed with the same groups and filters also applied.

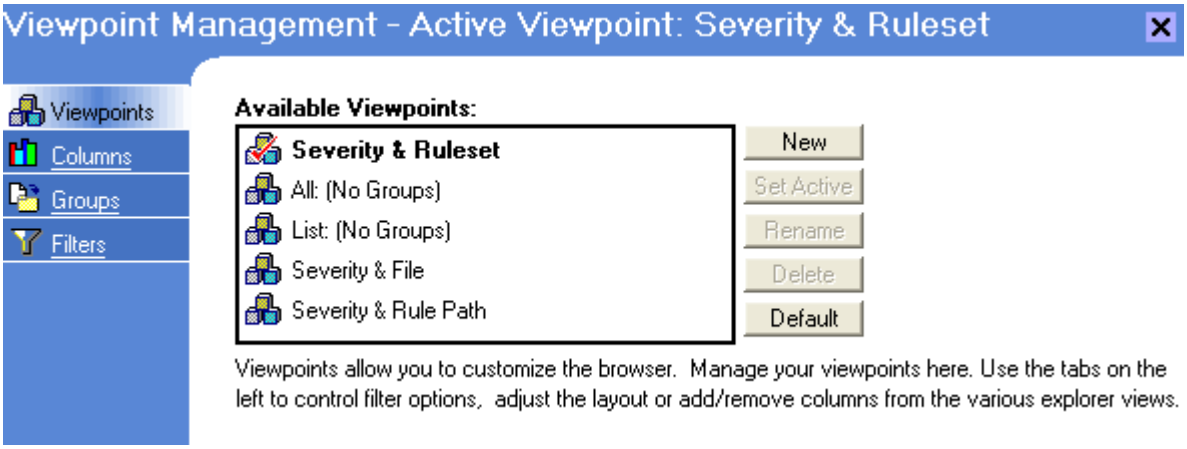
You can create and modify any number of viewpoints. These are automatically saved in your preferences.

You can display or hide the viewpoint manager window by choosing **Viewpoint Manager** from the **Results** menu or by using the  button or the  shortcut.

Several example viewpoints are defined:

- **Severity & Ruleset** — displays the code snippet, design unit name, hint, and message columns.
- **All: (No Groups)** — displays the results with all columns visible.
- **List: (No Groups)** — displays the design unit name, filename, hint, filename, leaf filename, library, line number, message, rule category, rule name, rule severity, ruleset and scope columns.
- **Severity & File** — displays the code snippet, hint, and message columns.
- **Severity & Rule Path** — displays the Severity & RulePath, library/design unit/scope, message, line number, code snippet and hint.

The default *Severity & Ruleset* viewpoint (indicated by the icon and name in bold font) is shown as active in the following example:



You can make a viewpoint active by using the Set Active button or you can create a new editable viewpoint by using the New button.

Note

When you create a new viewpoint it is automatically made active and given the same initial properties as the previous active viewpoint.

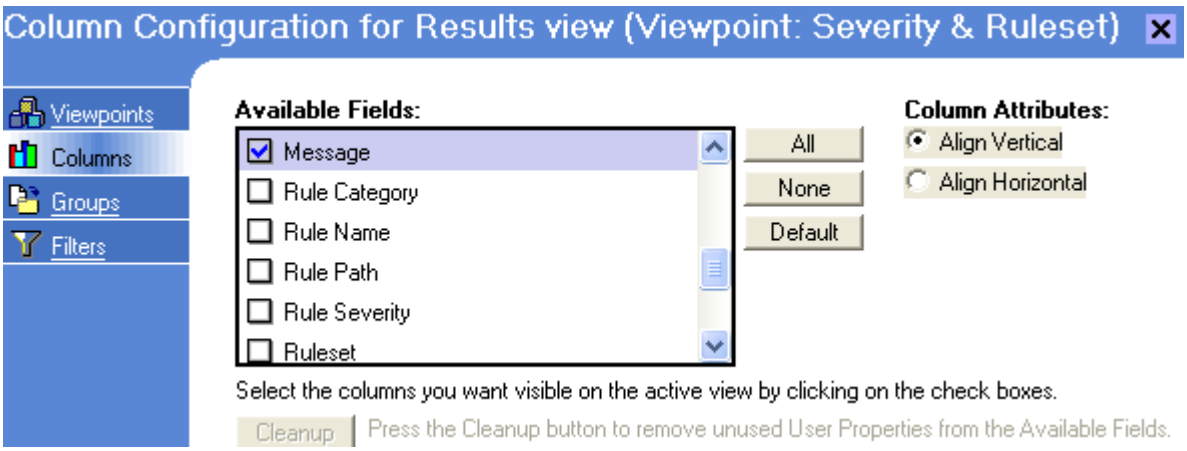
You can set the default viewpoint using the **Default** Default button.

You can delete the selected viewpoint using the Delete button or change the name of a viewpoint using the Rename button.

You can revert to the previously active viewpoint by choosing **Revert Viewpoint** from the **Results** menu or by using the button or shortcut.

Changing the Displayed Columns

You can choose the **Columns** option to display the column management window:



You can select from a list of available columns which includes:

- **Code Snippet** displays the code fragment containing the rule violation.
- **Design Unit Name** is the design unit containing a source design object.
- **File and Line** displays the leaf filename and line number.
- **Filename** is the full path for the physical file system object.
- **Full Message** is the full message text including the leaf filename, rule severity, rule name, rule category and message.
- **Full Message with Rule Path** is the full message text (including the leaf filename, line number, rule severity, message) plus the rule path.
- **Hint** is the suggested corrective action.
- **Leaf Filename** is the leaf name for the physical file system object.
- **Library** is the library containing the design object.
- **Library, Design Unit and Scope** displays the library name, design unit name and scope in the same cell.
- **Line Number** displays the line number within the file.
- **Message** is the text string describing the error, warning or note, or other user-defined severity levels.
- **Rule Category** is the category for the base rule.
- **Rule Name** is the name of the base rule.
- **Rule Path** is the full hierarchical name of the configured rule including the ruleset name.

- **Rule Severity** identifies whether the rule violation is an error, warning or note.
- **Ruleset** is the name of the ruleset.
- **Scope** is the type of object (Architecture, Configuration, Entity, File, Module or Package).
- **Severity and Rule Path** displays the severity and rule path.
- **Severity, Ruleset and Rule** displays the severity, ruleset and rule name in one cell.
- **Type** displays whether the violation is primary or associated.

The columns are displayed in the order they are added.

You can also change the displayed columns by checking options in the **Select Columns** cascade of the popup menu which is displayed if you click the 

mouse button over any column heading.

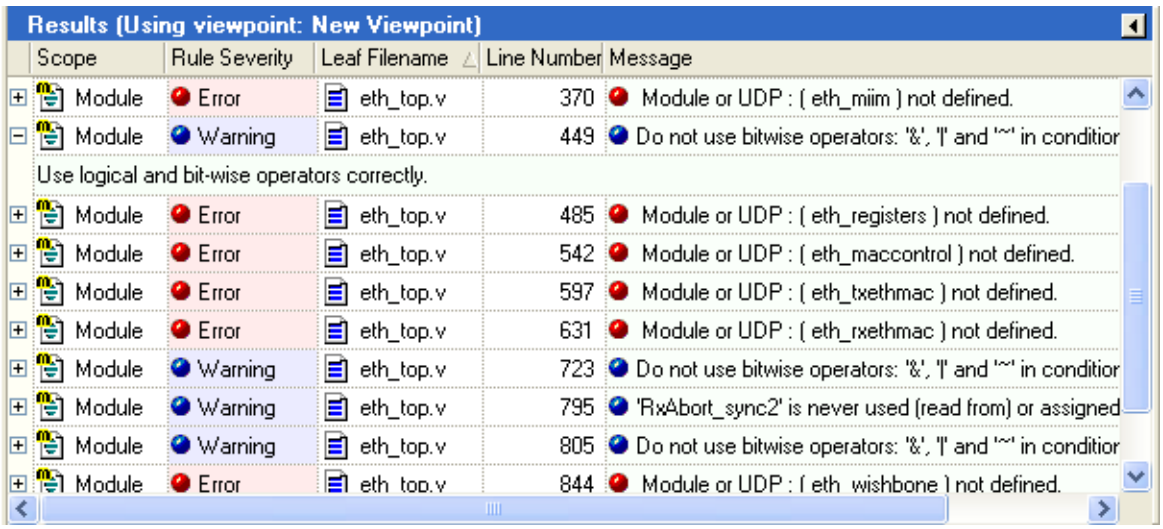
There is an attribute associated with each column in the list of available fields which controls how each column is displayed. This attribute defaults to **Align Vertical** for most of the columns. However, the attribute for the *Code Snippet* and *Hint* columns default to **Align Horizontal**.

Note

The column attributes control is not displayed until you select a column name in the list of available fields.

When the **Align Horizontal** attribute is set, the column is displayed in a separate row which can be expanded or collapsed using the ⊞ and ⊟ icons.

For example, the following picture shows an analysis for the *eth_top* design unit in the *Ethernet* example library (from HDL Designer Series) using a new viewpoint with columns displaying the scope, rule severity, leaf filename, line number, message and hint:



Note

It may be necessary to use the horizontal scroll bar if a viewpoint includes more columns than can be displayed in the window.

You can use the All button to select all available columns, None button to unselect all columns or Default button to reset the default columns.

You can change the width of the columns by dragging the sashes between each column or automatically resize a column to fit its contents by double-clicking on the sash.

Changing the Column Display Order

You can change the column display order by dragging the column header with the Left mouse button.

Changing the Object Row Display Order

You can re-order rows in ascending or descending order for the selected column by choosing **Sort Ascending** or **Sort Descending** from the popup menu when the cursor is over any column header.

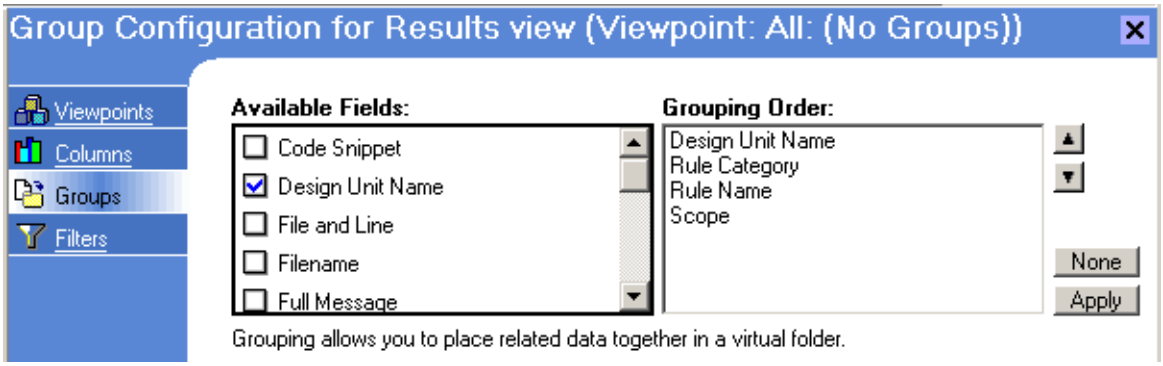
The new sort order is indicated by a ▲ or ▼ indicator in the column header.

Note

You can quickly toggle the existing order by clicking the Left mouse button in any column header.

Grouping Design Objects

You can choose the **Groups** option to set up groups for the current viewpoint:



You can choose to group by any of the column name fields listed in "Changing the Displayed Columns" on page 76.

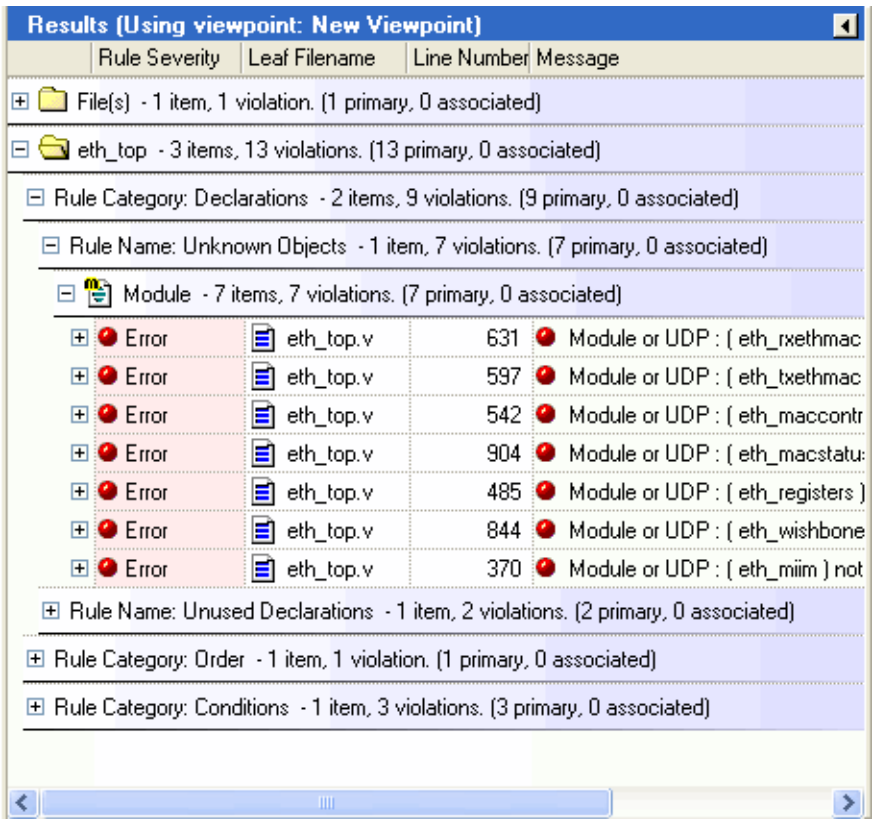
If you choose to group by a column which is not currently displayed it is added to the list of displayed columns. You can use the None button to unselect all groups.

The specified grouping is applied to the results tab when you use the Apply button.

If you choose more than one column to group by, the groups are nested in the order that you selected them. However, you can use the ▲ and ▼ buttons to set the grouping order and use the Apply button to apply the modified order to the table.

You can use the ⊞ icons to expand any group and reveal the objects it contains.

For example, the following picture shows the results from an analysis of the *eth_top* example from the *Ethernet* library (from HDL Designer Series) grouped by design unit name, rule category, rule name and scope:



You can also group design objects by choosing **Group by this column** from the popup menu for a column header.

You can choose **Ungroup** from the popup menu to remove all column groups.

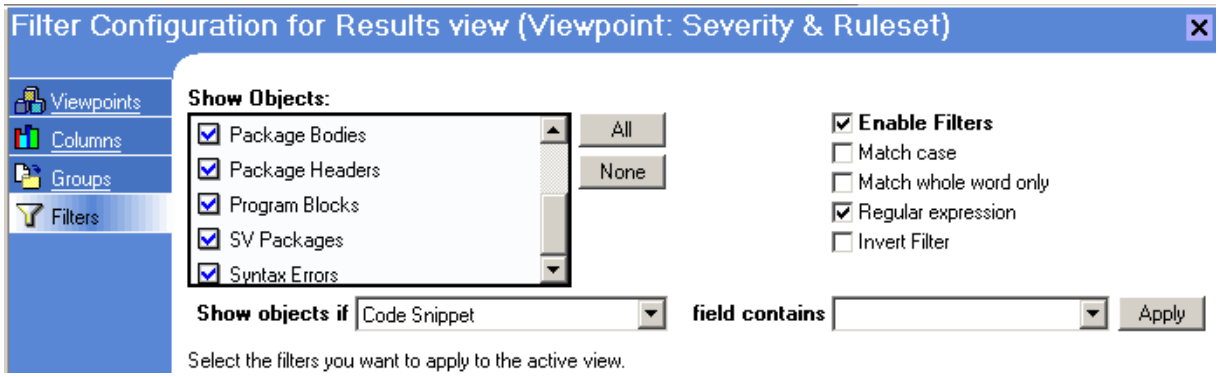
Filtering Results

DesignChecker is pre-configured with the following standard filters:

- Architectures
- Classes
- Configurations
- Entities
- Files
- Interfaces
- Modules

- Package Bodies
- Package Headers
- Program Blocks
- SV Packages
- Syntax Errors

Additionally you can use the **Filters** option to set up filters for the current viewpoint:



You can use the All button to select all the filters or None button to unselect all filters.

You can also filter the list by selecting one of the column fields from the dropdown list and entering a text string to show only those objects in the selected field which contain the specified string.

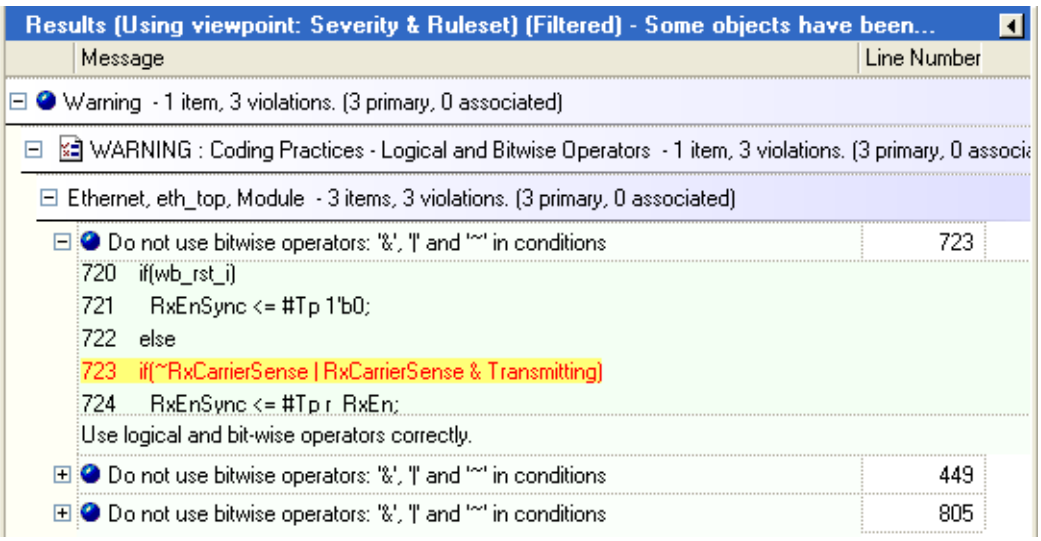
The filter can be a simple string match (including wildcards such as *acc**) or you can set options to match case, match whole word only or match a regular expression.

Note

Recently entered filter expressions can be selected from a dropdown list.
A list of supported regular expressions can be displayed by choosing **Regular Expressions** from the **Help** menu.

The displayed objects are filtered to show only objects that match the specified pattern.

For example, the following picture shows the results for the *eth_top* example (from HDL Designer Series) filtered for the text string *bitwise operators* in the *Message* column with the show modules object filter selected:



You can enable or disable the selected filter by setting or unsetting the **Enable Filters** option.

Use the **Apply** Apply button to apply your filter settings.

Note

The columns, groups and filters available in DesignChecker might change according to the interface tool you are using.

Adding a Viewpoint Shortcut

You can add a shortcut to any viewpoint by dragging its name from the viewpoint management window onto the Viewpoints group in the shortcut bar or remove a viewpoint shortcut by choosing **Remove** from the popup menu in the shortcut bar.

You can re-order shortcuts in the shortcut bar by dragging a shortcut to the required position.

The active viewpoint is identified in the viewpoint management window and shortcut bar by a red check mark overlay. For example, the following shortcut shows that the *Severity & File* viewpoint is active:

You can set the active viewpoint by clicking on a shortcut to change the viewpoint currently shown in the results tab.

You can access the Viewpoints shortcuts in DesignChecker at any time by choosing the Viewpoints Shortcut Bar.

Disabling and Enabling Checks

Checking and updating code is an iterative process. At times you will need to turn on or off certain checks within DesignChecker, without wishing to change either your user policies or rulesets. You can do this easily from the DesignChecker Results tab.

You can disable the rule corresponding to the selected result row by choosing **Disable Rule For** from the **Results** or popup menu, and then from the cascade menu, you can choose to disable the rule on the level of the policy, design unit or source file.

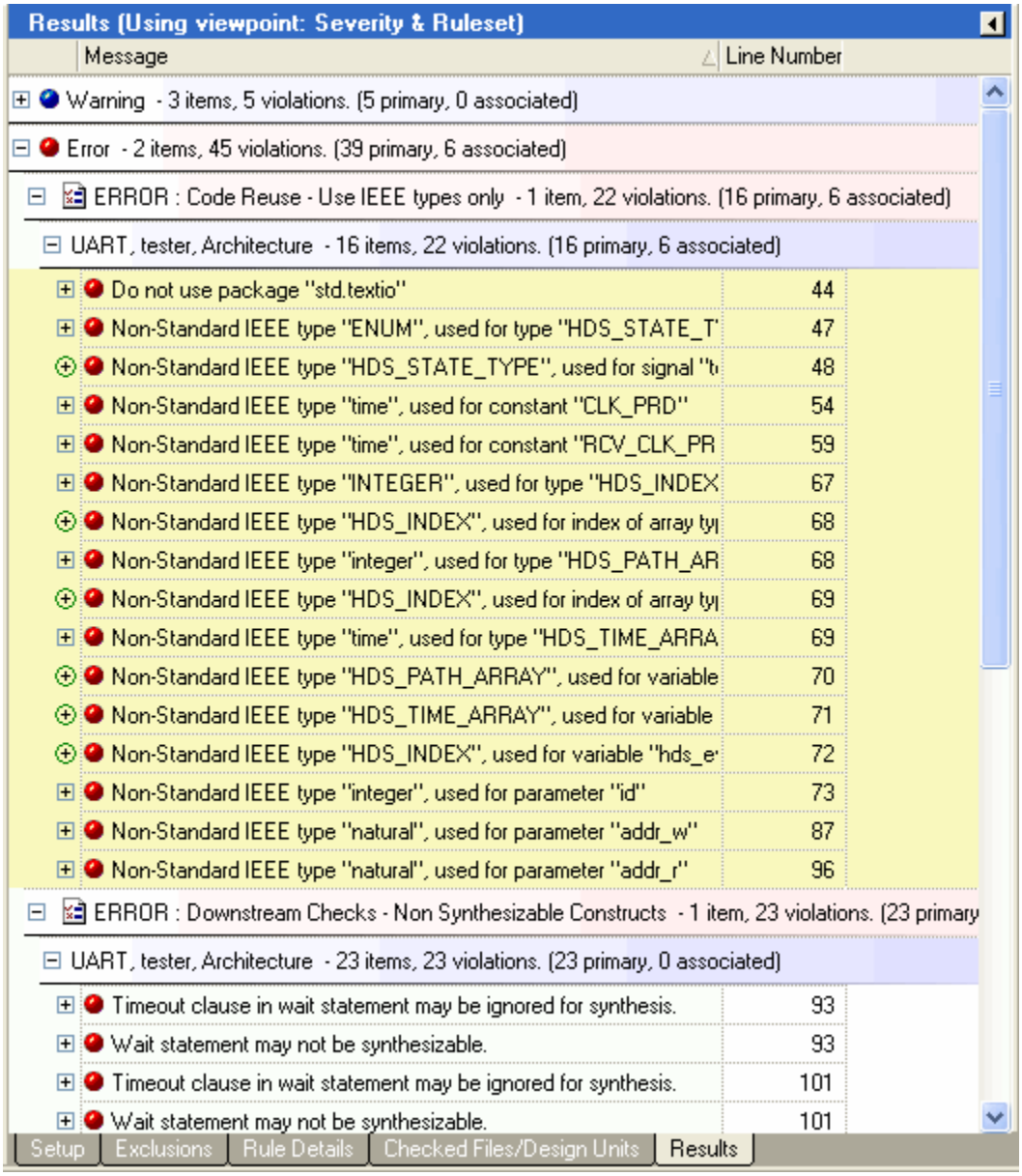
In the same way, you can enable the rule corresponding to the selected result row by choosing **Enable Rule For** from the popup menu, and then from the cascade menu, you can choose to enable the rule on the level of the policy, design unit or source file.

You can disable the ruleset corresponding to the selected result row by choosing **Disable RuleSet For** from the **Results** or popup menu, and then from the cascade menu, you can choose to disable the ruleset on the level of the policy, design unit or source file.

In the same way, you can enable the ruleset corresponding to the selected result row by choosing **Enable RuleSet For** from the popup menu, and then from the cascade menu, you can choose to enable the ruleset on the level of the policy, design unit or source file.

Note that the content of the popup menu may differ according to the interface tool you are using with DesignChecker. You can refer to the integration documents available through the **Help** menu of the interface tool.

By using the above explained methods to disable checks, the results relevant to the checks you disable will be highlighted in yellow as shown in the following figure.

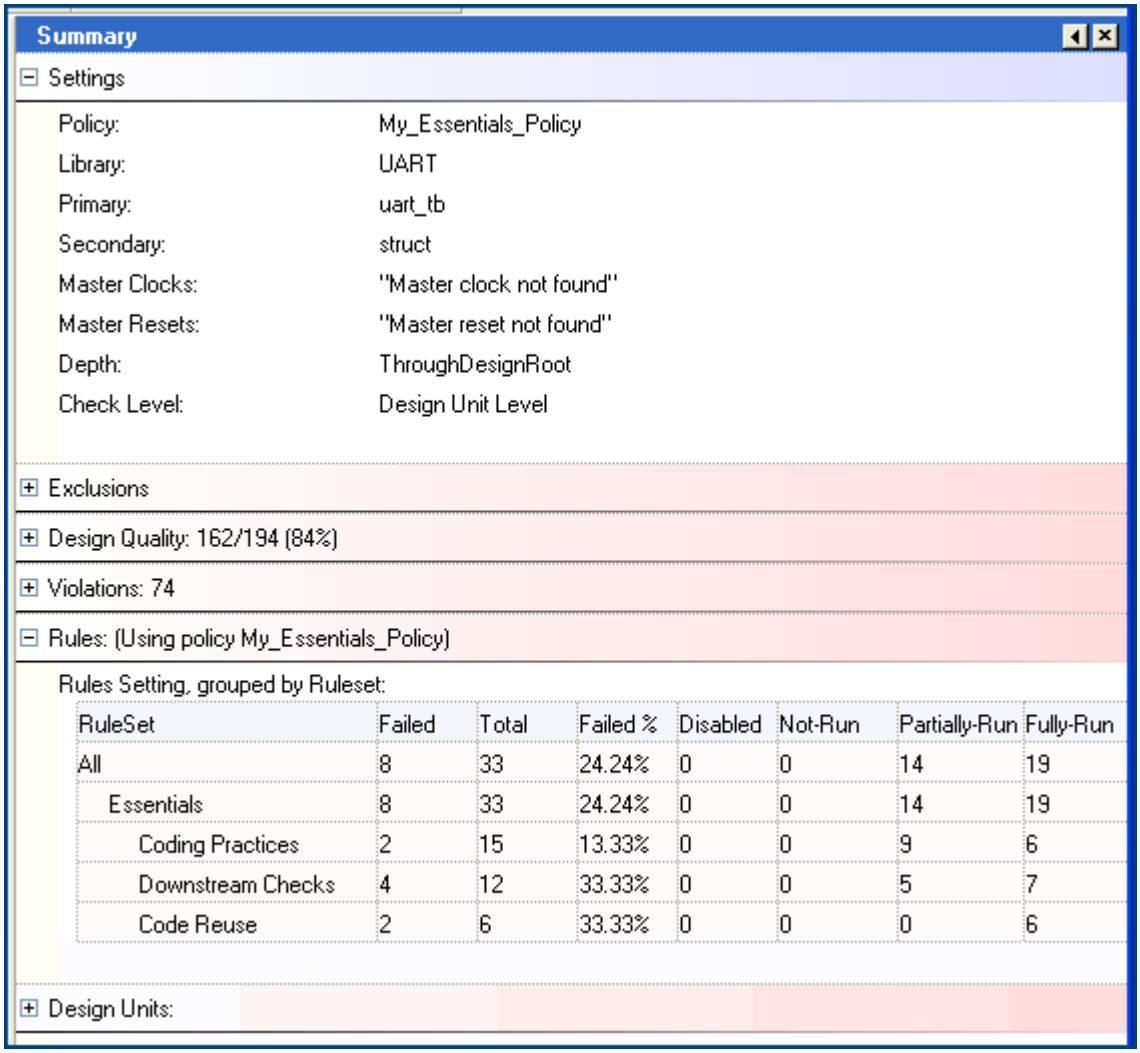


If you wish to remove the highlighted results of the disabled rule/ruleset, you can do so by selecting **Results > Remove Highlighted Results** or by clicking **Remove Highlighted Results**  in the shortcut bar on the left hand side. By doing that, the results will not be displayed in the Results tab.

Any rules or rulesets you disable for design units or source files are displayed in the Exclusions tab.

The Results Summary Pane

The results summary pane appears by default once DesignChecker has been run. This can be displayed or hidden by setting or unsetting **Summary Window** in the **View** menu (or hidden by unsetting this option in the popup menu when the window is displayed). In addition to providing an overview of the DesignChecker results, you can use the Summary pane to rapidly move between viewpoints by double-clicking on any item listed in the Summary pane tables. For example, double-clicking on *Warnings* in the Violations section, changes the viewpoint to show only warning violations.



The results summary pane displays an overview of the results in tabular form under the following headings:

- **Settings:** This displays information related to the design that has been analyzed such as the policy that has been used, the name of the design that has been analyzed, and so on.

- **Exclusions:** This section shows information on the exclusion settings that affected the latest analysis results.
- **Design Quality:** This displays the results of the design quality analysis. If the Design Quality Metric feature is not activated, then this section does not display any information and only indicates that "Analysis is turned off." For further information on this section, refer to "Reporting Quality Scoring" on page 87.
- **Violations:** Displays the number of violations for each of the error, warning and note severities, or other user-defined severity levels. Also shows the number of violations and the percentage in each scope.

  You can right-click on any item and select **Show Results** from the popup menu. By that, only the results pertinent to that item are displayed in the Results pane.
- **Rules:** Shows the name of the current default policy used for the analysis and shows information related to each ruleset.

  You can right-click on any ruleset and select **Show Results** from the popup menu. By that, only the results pertinent to that ruleset are displayed in the Results pane. By right-clicking on a ruleset and selecting **Show RuleSet** from the popup menu, the Rule Details tab will be opened displaying the corresponding ruleset and its content. Furthermore, you have the ability to enable or disable a ruleset by right-clicking on the ruleset's row and selecting **Enable RuleSet for Policy** or **Disable RuleSet for Policy**; if the rule is currently enabled then only Disable RuleSet for Policy will be activated and vice-versa.
- **Design Units:** Displays various information on each design unit type involved in the analysis.

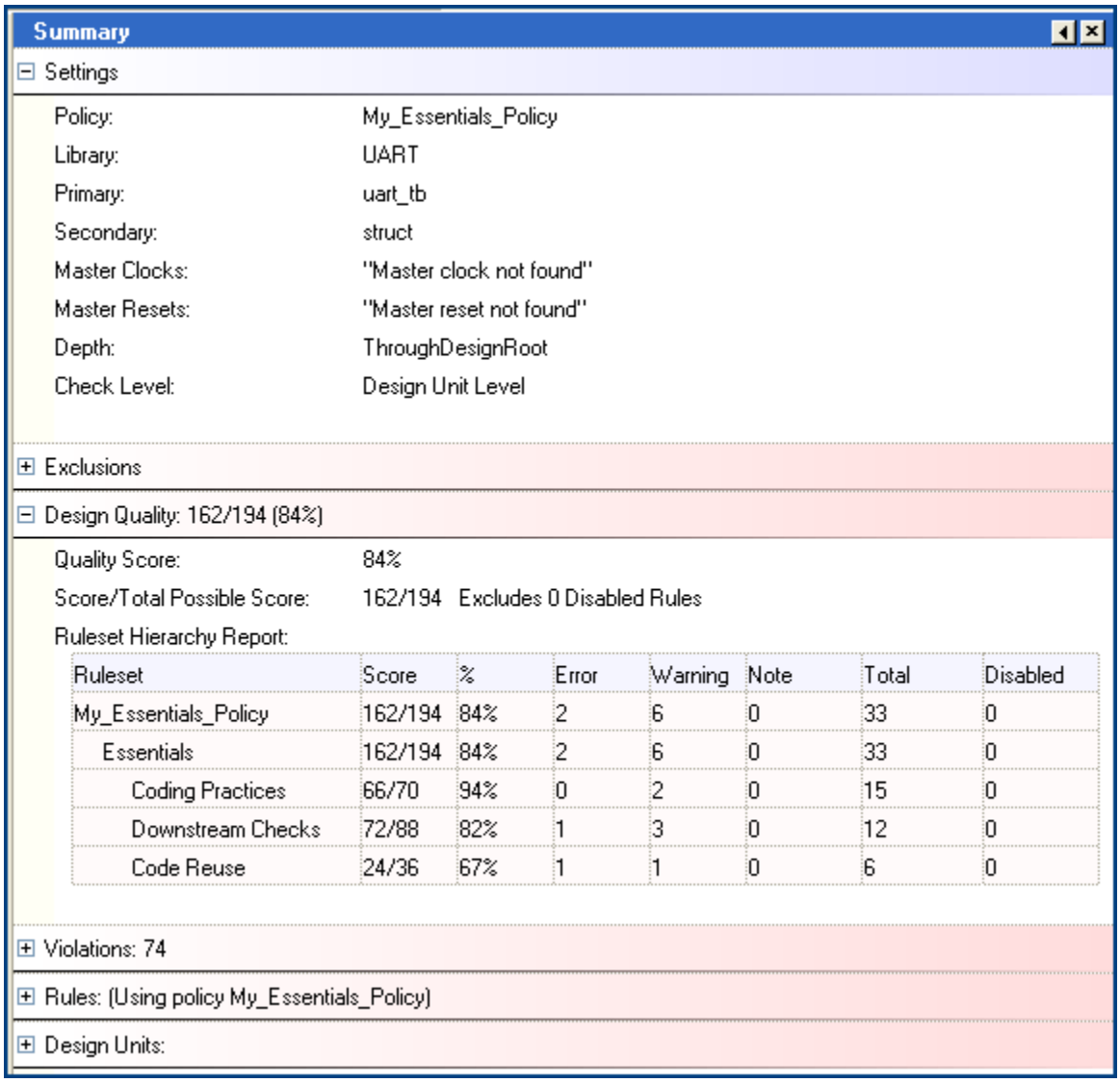
  You can right-click on any design unit type and select **Show Results** from the popup menu. By that, only the results pertinent to that item are displayed in the Results pane.

Note

You can collapse or expand the summary tables by using the ⊟ or ⊞ icons.

Reporting Quality Scoring

After running an analysis for your design, the Results tab is opened showing the quality scoring results in the Summary pane.



Ruleset	Score	%	Error	Warning	Note	Total	Disabled
My_Essentials_Policy	162/194	84%	2	6	0	33	0
Essentials	162/194	84%	2	6	0	33	0
Coding Practices	66/70	94%	0	2	0	15	0
Downstream Checks	72/88	82%	1	3	0	12	0
Code Reuse	24/36	67%	1	1	0	6	0

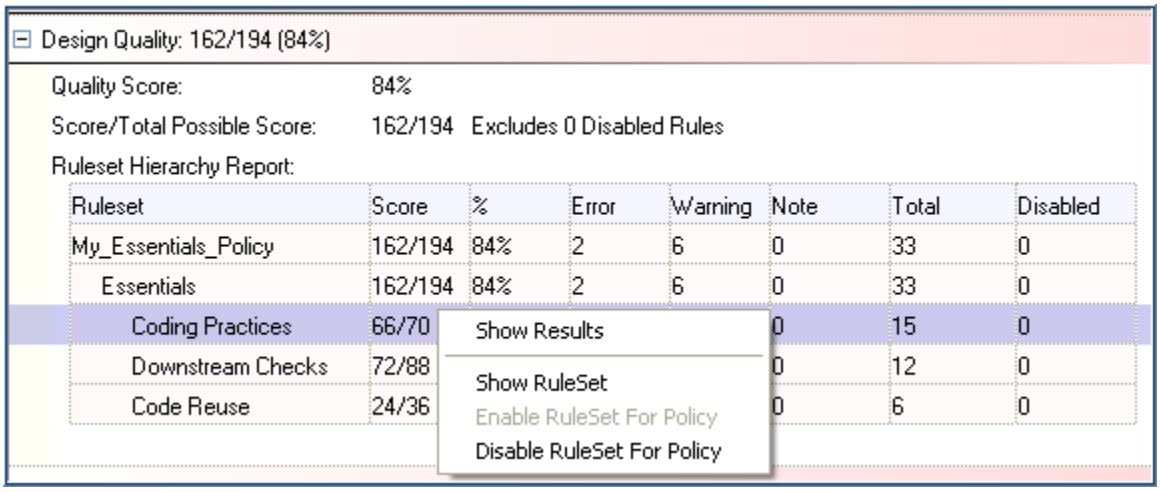
In the Design Quality section, you are given the details on the quality scoring calculation:

- **Quality Score** — the percentage of the code quality.
- **Score/Total Possible Score** — the total actual score and the total possible score; in addition to that, the number of disabled rules included (in case the Include Disabled Rules option is set in the Design Quality Options dialog box) or the number of disabled rules excluded (in case the option is not set).
- **RuleSet Hierarchy Report** — a hierarchy of the rulesets within the policy. This section shows the score of each ruleset, the quality percentage, the number of violated rules for

each severity level, the total number of configured rules in each ruleset, and the number of disabled rules in each ruleset.

You can filter the violations displayed in the Results pane by a specific ruleset. This is achieved either by double-clicking on that ruleset in the RuleSet Hierarchy Report table, or by right-clicking on the ruleset and selecting **Show Results**. By that, only the results pertinent to that ruleset are displayed in the Results pane. This can also be applied on the level of the policy.

Furthermore, you have the ability to enable or disable a ruleset directly through the RuleSet Hierarchy Report table by right-clicking on the ruleset and selecting **Enable RuleSet for Policy** or **Disable RuleSet for Policy**; if the rule is currently enabled then only Disable RuleSet for Policy will be activated and vice-versa.



By right-clicking on a ruleset and selecting **Show RuleSet** from the popup menu, the Rule Details tab will be opened displaying the corresponding ruleset and its content.

Note

If quality analysis is disabled in the Design Quality Options dialog box, then the Design Quality section in Summary pane will not display the above mentioned data. Instead, it will only display the statement “Analysis turned off.” Note that you can open the Design Quality Options dialog box by right-clicking inside the Design Quality section and selecting **Design Quality Options**.

Cross-referencing Results

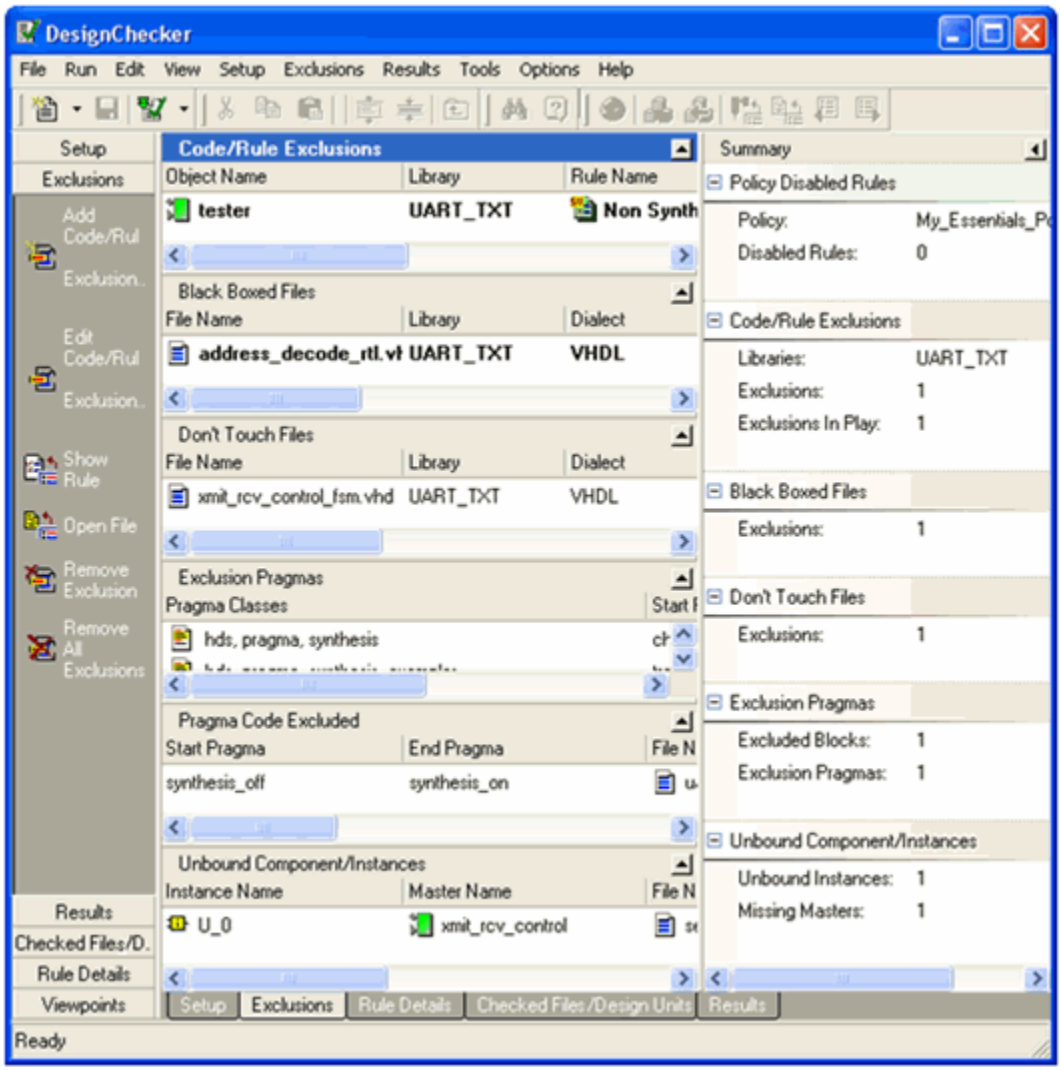
You can easily cross-reference between your DesignChecker results and your source code by right-clicking on any code snippet in the DesignChecker Results tab and selecting **Open HDL**/**Open Source** from the popup menu. By doing that, the text editor corresponding to the interface tool you are using will be invoked showing the source code in which the violation occurs.

With some tools, you will be able to cross-reference your results with graphical source views.

Refer to the integration documents available through the **Help** menu of the interface tool for further information.

The Exclusions Tab

Once you have run DesignChecker, the Exclusions tab is available from which you can review the exclusion settings related to the last run. The Exclusions tab consists of several panes that reflect the exclusion types used with the interface tool. For example, in case of using HDL Designer Series with DesignChecker, this tab will display code/rule exclusions, black box exclusions, Don't Touch exclusions, pragma exclusions and so on.



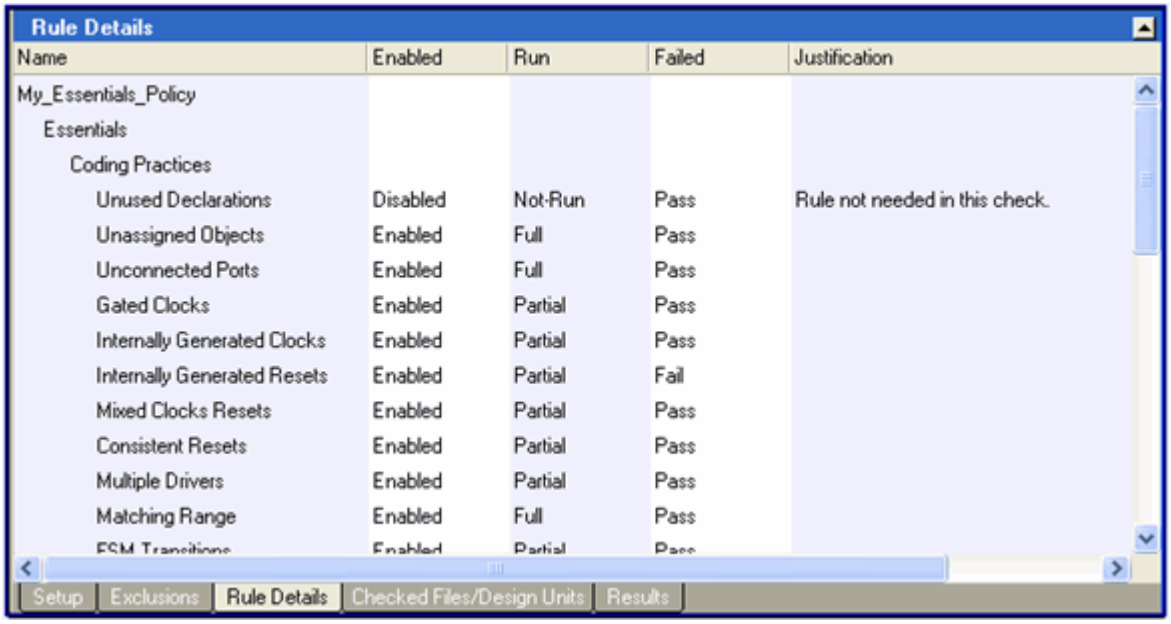
You can perform some operations through the Exclusions tab that differ according to the exclusion type. For example, you can add or remove certain exclusion types directly through the Exclusions tab, you can cross-reference to the source black-boxed or Don't Touch files, and so on.

The Exclusions tab also contains a summary pane on the right including summarized data on the exclusion settings applied in the latest analysis.

For more details on the content of the Exclusions tab, refer to the integration documents available through the **Help** menu of your interface tool.

The Rule Details Tab

This tab displays the policy applied to the latest DesignChecker analysis. It shows a list of the rulesets in the policy along with the pertinent configured rules.



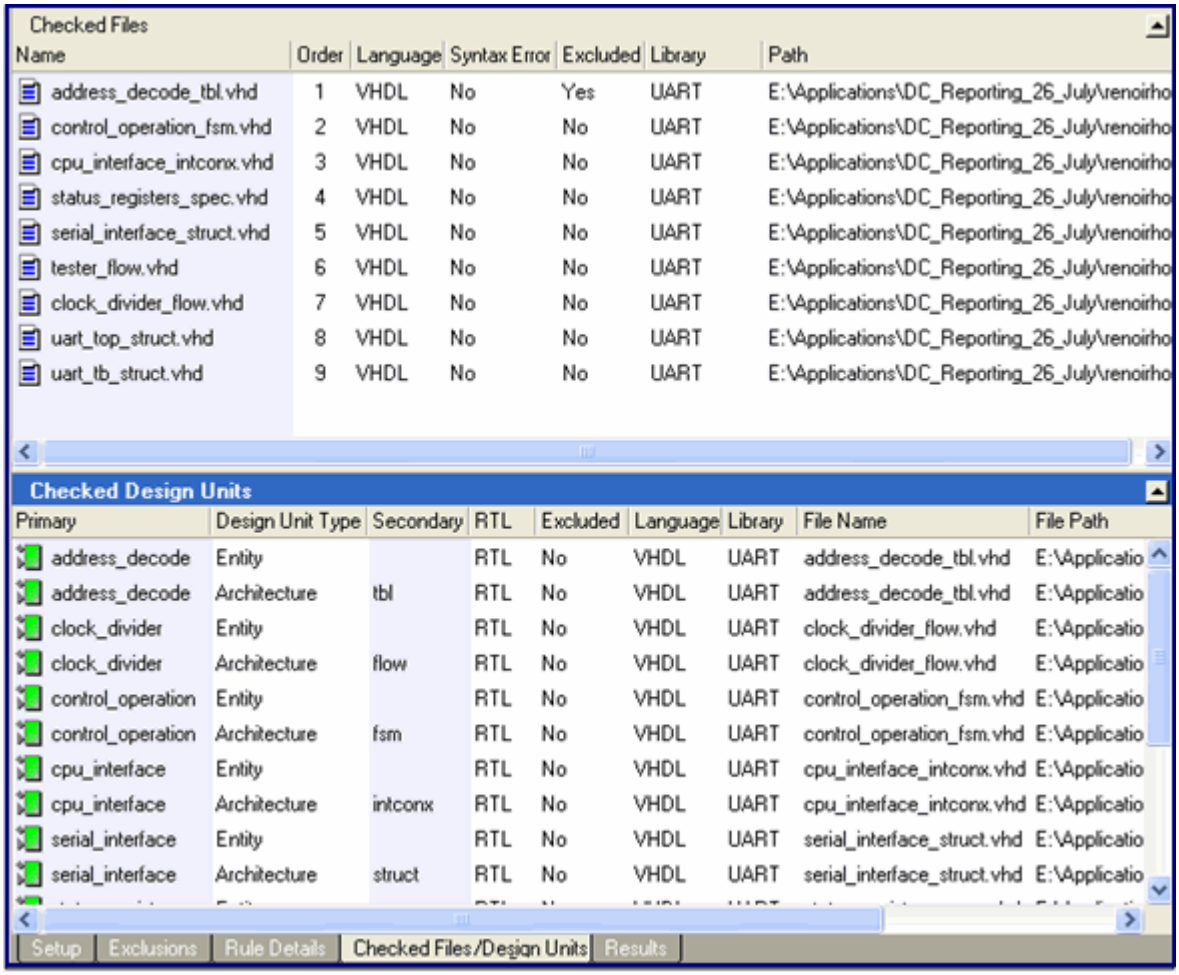
The Rule Details tab contains the following columns:

- **Name** — shows the name of the policy, ruleset or configured rule.
- **Enabled** — shows whether the rule is enabled or disabled.
- **Run** — shows whether the rule was fully run (run on the entire design), partially run (run on part of the design), or not run.
- **Failed** — shows whether the rule has been violated (Fail) or not (Pass).
- **Justification** — shows the justification for disabling the rule or ruleset.

Double-clicking on any rule will open the Setup tab with the same rule highlighted, or you can right-click on any rule and select **Show Rule/RuleSet** from the popup menu. The same applies to rulesets.

The Checked Files/Design Units Tab

This tab shows the files and design units that have been analyzed in the latest DesignChecker analysis. This tab contains two panes: the Checked Files pane and the Checked Design Units pane. The Checked Files pane shows all the files that have undergone the last analysis, and similarly, the Checked Design Units pane shows all the design units that have undergone the last analysis.



If you run file level checking, then the analyzed file will be displayed in the Checked Files pane and only the design units related to that file will be displayed in the Checked Design Units pane. Likewise, if you run design unit level checking, the analyzed design unit will be displayed in the Checked Design Units pane, and the file to which the design unit belongs will be displayed in the Checked Files pane.

Each pane constitutes of columns showing information on the file or design unit.

Columns in the Checked Files pane:

- **Name** — shows the name of the analyzed file.

- **Order** — shows the order of the file in analysis.
- **Language** — shows whether the language of the file is VHDL, Verilog or System Verilog.
- **Syntax Error(s)** — indicates the existence of syntax errors in the file.
- **Excluded** — shows whether the file is excluded from analysis or not.
- **Library** — shows the name of the library to which the file belongs.
- **Path** — shows the location of the file on the hard disk.

Columns in the Checked Design Units pane:

- **Primary** — shows the primary name of the design unit.
- **Design Unit Type** — shows the type of the design unit. For example: entity, architecture, SV Package, Interface, and so on.
- **Secondary** — shows the secondary name of the design unit.
- **RTL** — shows the RTL status of the design unit: RTL, Non-RTL or not applicable.
- **Excluded** — shows whether the design unit is excluded from analysis or not.
- **Language** — shows whether the language is VHDL, Verilog or System Verilog.
- **Library** — shows the name of the library to which the design unit belongs.
- **File Name** — shows the name of the file to which the design unit belongs.
- **File Path** — shows the location of the file to which the design unit belongs.

Note

Note that the panes in the tab may differ according to the interface tool you are using with DesignChecker. Some tools use both the Checked Files and Checked Design Units pane, other tools may use only the Checked Files pane.

Exporting Results

You can export various information on the latest DesignChecker analysis. The Export Results dialog box allows you to generate different types of reports. This dialog box enables you to generate separate reports on the latest results, the summary, the exclusion settings applied to the analysis, details on the rules used in the analysis, in addition to information on the files/design units that have been analyzed.

Procedure

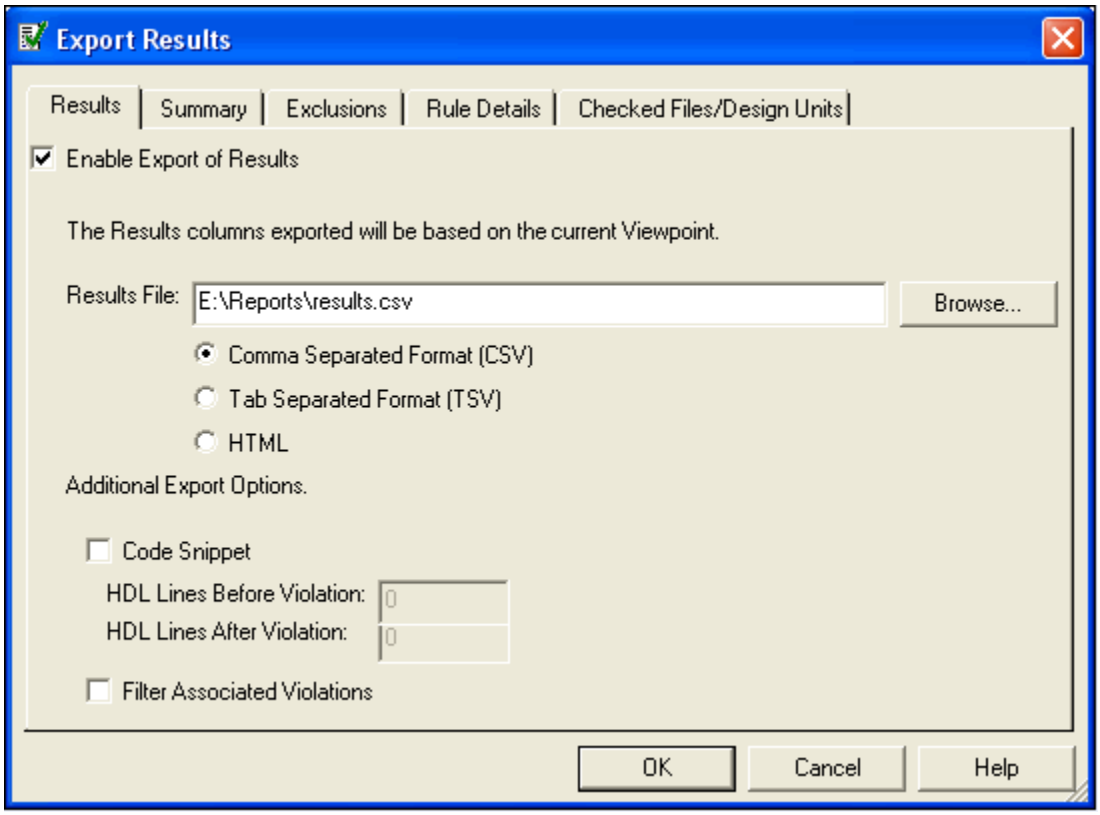
1. Open the Export Results dialog box by choosing **File > Export Results** or click  in the toolbar. Note that you will not be able to open the Export Results dialog box if the **Setup** tab is active.
2. Open the tab corresponding to the information you wish to export: **Results**, **Summary**, **Exclusions**, **Rule Details**, or **Checked Files/Design Units**.
3. According to the tab you opened, enable the export of its corresponding information.

   For example, if you open the **Results** tab, you can enforce exporting the analysis results by setting the option “Enable Export of Results”.

   Note that if you enable the export, then you have to configure the remaining options in the **Results** tab. If you wish to disable the export, then you can ignore the remaining options and move on to other tabs (such as **Summary**, **Exclusions**, and so on).
4. Specify the path of the file(s) to which you will export the information. For example, if you have enabled the export of results and summary, then you have to specify the path of these two reports.
5. Select the type of the file: **Comma Separated Format (CSV)**, **Tab Separated Format (TSV)**, **HTML**.

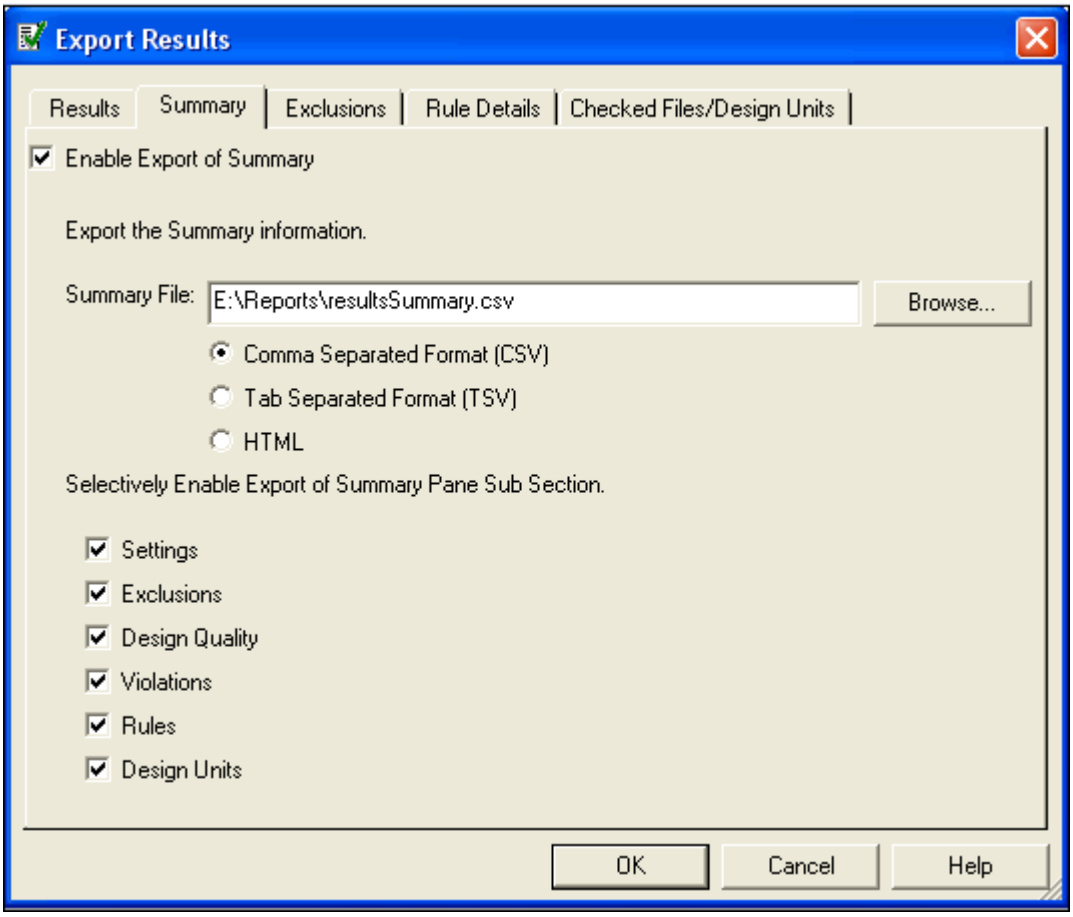
   CSV format is written with the file extension.*csv*, TSV format with the file extension *.txt* and HTML format with the extension *.htm*. These file extensions are normally registered on a Windows system to open CSV files in Microsoft Excel, TSV files in your default text editor and HTML files in your default web browser.
6. Configure the options related to the information you are exporting as follows:

- **Results Tab:**



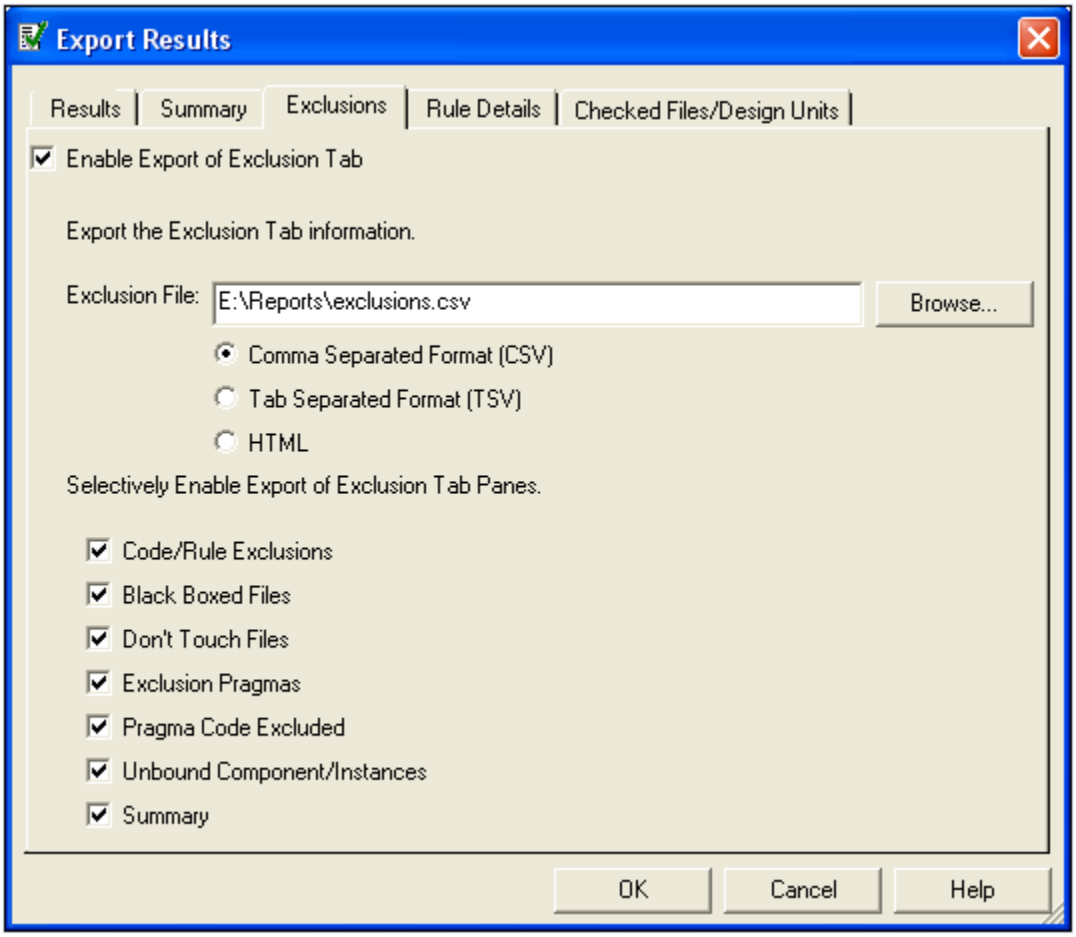
  - Specify whether you want to display the code snippet in which the violation occurred through the option Code Snippet. If selected, you can also specify the number of code lines before and after the violation line that you want to include in the report.
  - Specify whether you want to exclude any associated violations from the report through the option Filter Associated Violations.

- **Summary Tab:**



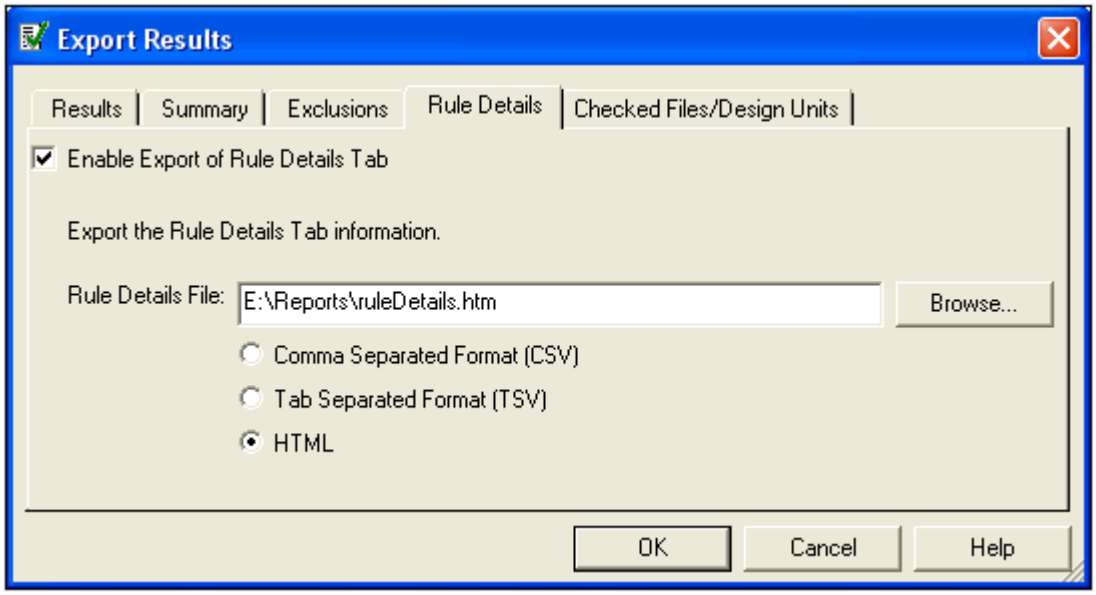
  - The Summary report displays all the content of the Summary pane by default. You can specify certain sections of the Summary pane to display: Settings, Exclusions, Design Quality, Violations, Rules, and Design Units. Refer to "The Results Summary Pane" on page 85 for more information on these sections.

- **Exclusions Tab:**



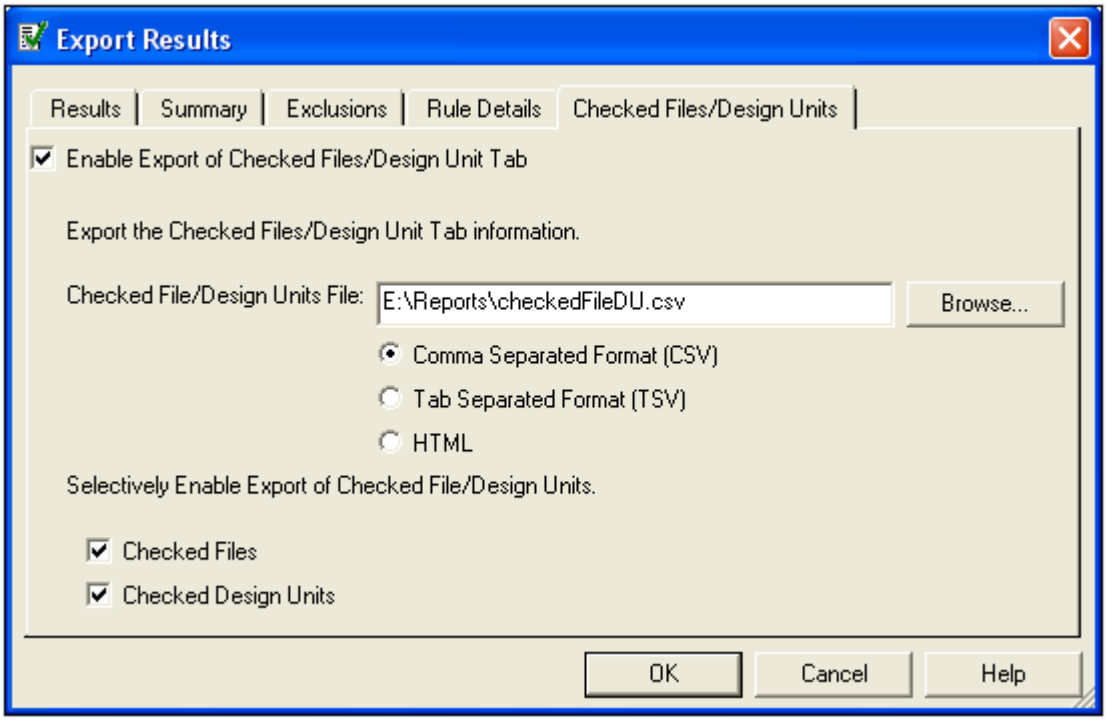
  - The Exclusions report displays all the content of the Exclusions tab by default. You can specify certain sections of the Exclusions tab to display: Code/Rule Exclusions, Black Boxed Files, Don’t Touch Files, Exclusion Pragmas (the number of pragma types used in the analyzed design), Pragma Code Excluded, Unbound Component/Instances, and Summary.

- **Rule Details Tab:**



  - The Rule Details report does not require extra configuration.

- **Checked Files/Design Units Tab:**



  - You have to specify if you need a report on Checked Files or Checked Design Units, or both.

7. Click **OK** after setting the required options in each tab.

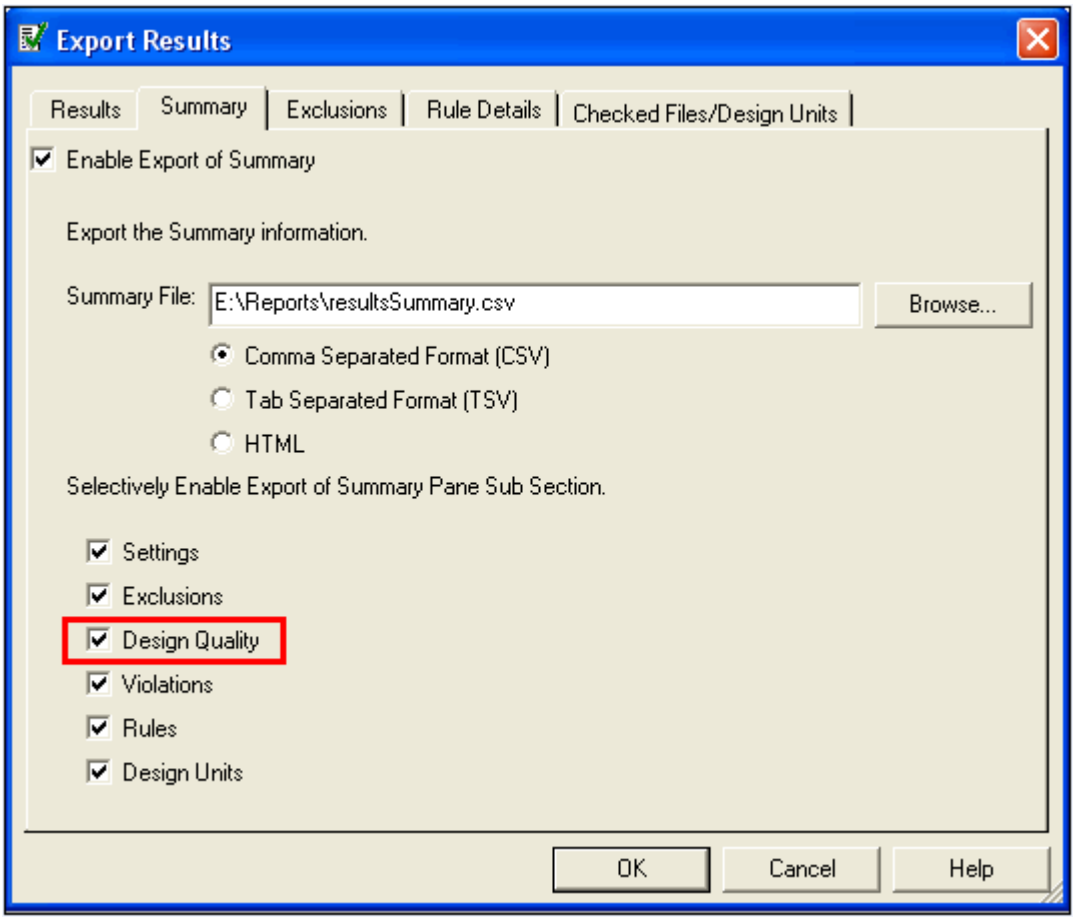
Consequently, the separate reports are generated in the specified paths. You are prompted to overwrite the files if they already exist at the specified locations.

Note

Note that the options within each tab might differ according to the interface tool you are using with DesignChecker.

Exporting Quality Reports

Using the Export Results dialog box, you can export a Summary report including the quality analysis results, but you have to make sure that the Design Quality option is selected as highlighted in the below figure. The exported report can be in one of the following formats: Comma-Separated Format (CSV), Tab-Separated Format (TSV), or HTML.



The following figure shows an excerpt of an exported Summary (in HTML format) including the Design Quality section:

Settings

Policy:	My_Essentials_Policy
Library:	UART
Primary:	uart_tb
Secondary:	struct
Master Clocks:	"Master clock not found"
Master Resets:	"Master reset not found"
Depth:	ThroughDesignRoot
Check Level:	Design Unit Level

Number of Exclusions

Policy Disabled Rules	0
Code/Rule Exclusions	0
Black Boxed Files	1
Don't Touch Files	0
Exclusion Pragmas	1
Pragma Code Excluded	61
Missing Masters	0
Unbound Instances	0

Design Quality: 162/194 (84%)

Quality Score:	84%
Score/Total Possible Score:	162/194 Excludes 0 Disabled Rules

Ruleset Hierarchy Report:

Ruleset	Score	%	Error	Warning	Note	Total	Disabled
My_Essentials_Policy	162/194	84%	2	6	0	33	0
Essentials	162/194	84%	2	6	0	33	0
Coding Practices	66/70	94%	0	2	0	15	0
Downstream Checks	72/88	82%	1	3	0	12	0
Code Reuse	24/36	67%	1	1	0	6	0

Appendix A
Supporting Synthesizable Designs

DesignChecker contains a number of rules which can only run on synthesizable designs. The section Rules Specific to Synthesizable Code provides a list of those rules which require the design to be complete and synthesizable first prior to analysis; this section is applicable only to RTL tools that interface with DesignChecker. On the other hand, the section SystemVerilog Support provides a list of all the synthesizable SystemVerilog constructs supported by DesignChecker.

Rules Specific to Synthesizable Code

The following rules can only analyze synthesizable code. On running DesignChecker, these rules can cause "synthesis errors" if they find unsynthesizable constructs in the design. Similarly, these rules can cause "elaboration errors" if the design is incomplete or if there are unsupported constructs; refer to "Understanding the Types of Violations" on page 74 for more information about synthesis and elaboration errors.

- Allowed Constructs
- Asynchronous Load Signals
- Asynchronous Reset Release
- Arithmetic Gate Report
- Blocking Nonblocking Assignments
- Case instead of if-else-if
- Case Statement Directives
- Case Statement Style
- Clock Declaration Style
- Clock Used As Data
- Combinatorial Feedback
- Consistent Clocks
- Consistent Resets
- Edge Trigger Control
- FSM Coding Style
- FSM Complexity
- Matching Range
- Memory Element Report
- Mixed Assignments
- Mixed Clocks Resets
- Multiple Clocks
- Multiple Resets
- Multiple Drivers
- Non-Static Ranges
- Non-Unrollable Loops
- Port Mapping
- Re-entrant Outputs
- Register Controllability
- Register IO
- Register Race
- Register Reset Control
- Related Combinatorial Logic

- FSM Report
- FSM State Encoding Style
- FSM Transitions
- Gated Clocks
- Inferred Elements Naming
- Initialization Assignments
- Input Port Assignments
- Internally Generated Clocks
- Internally Generated Resets
- Latch Inference
- Reset Logic Function
- Sensitivity List
- Snake Paths
- Sub-Program Body
- SV Always Block
- Undriven & Unused Logic
- Unique Names
- Unreachable Blocks
- Untested Edge Trigger
- Unused Declarations

Note

During analysis, the above rules will ignore any code inside initial blocks and *translate_off/on* or *synthesis_off/on* pragmas.

SystemVerilog Support

Some rules in DesignChecker support SystemVerilog designs. DesignChecker provides support for just the following SystemVerilog design units: Modules, Packages and Interfaces.

Note

All the base rules supporting SystemVerilog in DesignChecker are marked with a SystemVerilog overlay  .

This section lists the synthesizable SystemVerilog constructs and features supported by DesignChecker.

Table A-1. Supported SystemVerilog Constructs

Construct	Details
Literal Values	• Integer • Logic • Real • String of type Packed Array • Array • Structure Literals

Table A-1. Supported SystemVerilog Constructs (cont.)

Construct	Details
Data Types	• Integer (integer, int, shortint, longint, byte, logic, bit, reg) • Void • User-defined Types • Enumerations • Structures (Packed/Unpacked) and Unions (Packed) • Casting
Arrays	• Packed and Unpacked Arrays • Multi-dimensioned Arrays • Array Indexing and Slicing • Array Assignment • Arrays as Arguments
Data Declarations	• Constants • Signal Aliasing
Attributes	
Operators and Expressions	• Assignment Operators • Wild Equality and Inequality Operators

Procedural Statements and Control Flow	• Selection Statements
	• Loop Statements
	  • Do-While Loop
	  • Enhanced For Loop
	• Jump Statements
	  • Break
	  • Continue
	  • Return
	• Named Blocks
	• Event Control (iff Qualifier)

Processes	• Combinatorial Logic • Latched Logic • Sequential Logic

Tasks and Functions	• Void Functions
	• Argument Passing
	  • Default Argument Values
	  • Argument Passing by Name

Table A-1. Supported SystemVerilog Constructs (cont.)

Construct	Details

Hierarchy	• Port Declarations
	• Port Connection Rules
	• Module Instances
	  • Instantiation Using Implicit .name Port Connections
	  • Instantiation Using Implicit .* Port Connections

Interfaces	
System Tasks and System Functions	
Compiler Directives	• `define Macros

Appendix B
Further Understanding Design Checking Rule Behavior

In this appendix we will try to cover the different factors affecting the way DesignChecker rules operate. In doing so we aim to provide a deeper understanding of the violations produced in terms of reason of issuance and meaning.

The following topics will be explored:

1. Design Correctness and Synthesizability
2. Register and Control Signal Inference
3. Finite State Machines (FSMs)
4. Miscellaneous

These topics are mainly applicable to RTL tools that interface with DesignChecker such as HDL Designer Series and Visual Elite.

Design Correctness and Synthesizability

Introduction

For synthesis rules to run on a design, the design needs to be correct, i.e. the design should not contain any syntax or elaboration errors. When a syntax or elaboration error is encountered, synthesis rules will not be run on the entire design.

Designs can also contain non-synthesizable modules, which also results in synthesis errors. When a non-synthesizable module is encountered in a design, synthesis rules will not be run on that module only. However, design checking can still be carried out on the rest of the design.

Note

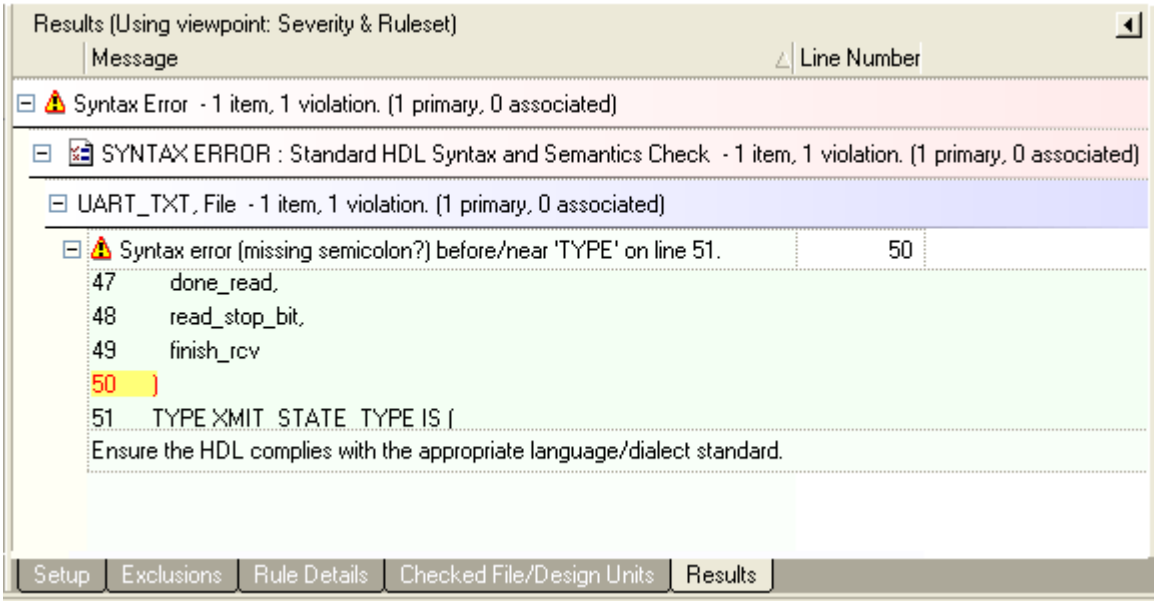
DesignChecker may not always be able to report all errors in the design in a single run and multiple runs may be required to make the design completely error-free.

All the different types of errors (syntax, elaboration and synthesis) are described in greater detail in this section.

Syntax Errors

These result when DesignChecker encounters syntactically incorrect code. In such cases, none of the rules run on the entire design, and only syntax error violations are displayed. These syntax errors need to be corrected before DesignChecker can perform design analysis.

The following figure shows syntax errors violations as displayed in the Results tab.



Elaboration and Elaboration Errors

Synthesis rules always operate on a fully elaborated design. Elaboration can loosely be described as a process by which overridden generic/parameter values in an instantiation are transferred to the instance master and appropriate port connections are established. Consider the following example:

```
module top (input in1, sel, output out1);
    middle #(.p1(7)) inst (.in1(in1), .sel(sel), .out1(out1));
endmodule
module middle (input in1, sel, output reg out1);
parameter p1 = 3;
    always @(in1 or sel)
    begin
        if(p1 == 3)
            out1 <= in1;
        else
            out1 <= sel;
    end
endmodule
```

In this case, when instance “inst” of middle is elaborated, the value 7 is propagated to the parameter p1. As a result, the condition (p1 == 3) will not be true, even if it seems to be the case if middle is seen in isolation.

Failure to elaborate a design results in elaboration errors. Some common causes of elaboration errors are as listed below:

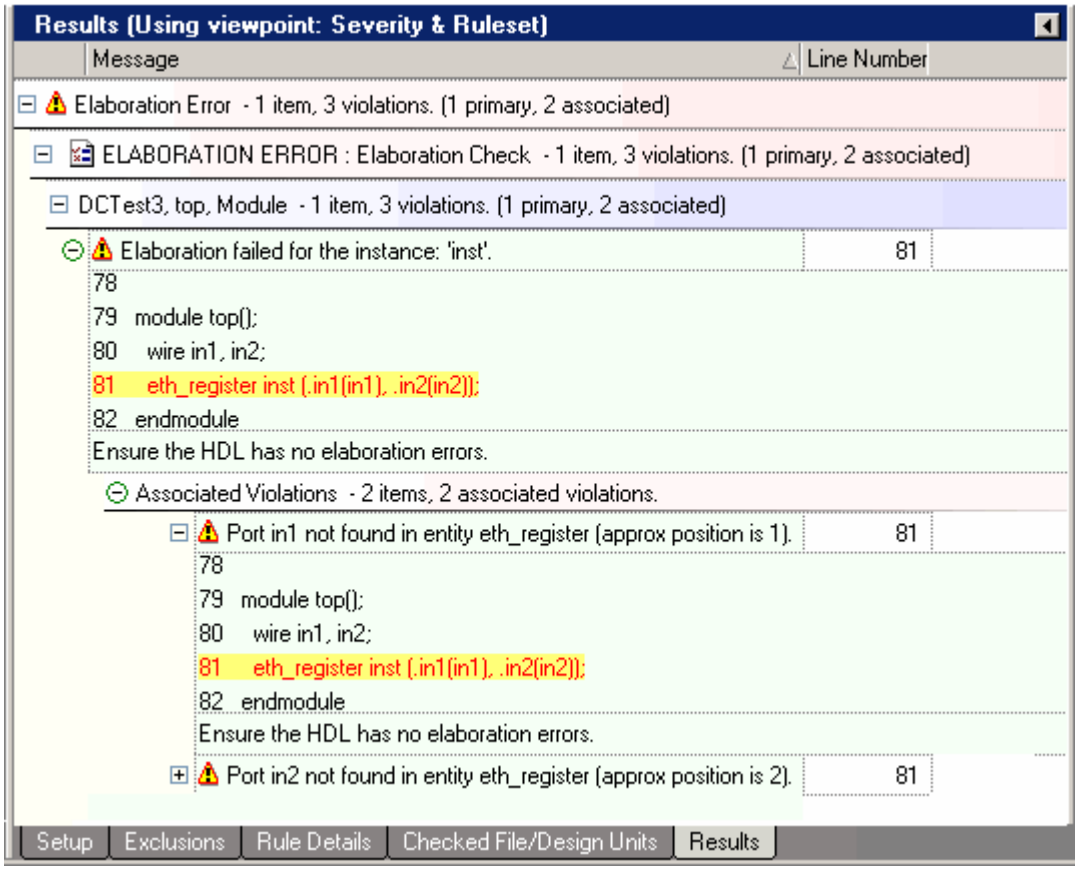
- Name/type mismatch in the port/generic map of an instantiation and the corresponding port/generic in the instance master.
- Incorrect direction in the component of an instance and the corresponding port in the instance master.

In addition to the above mentioned causes, elaboration errors are also produced in the following scenarios:

- Top-level generic values not specified.
- Top-level port sizes not specified.

Whenever DesignChecker configured to run some synthesis rules encounters an elaboration error, none of the synthesis rules will be run on the design and elaboration error violations are displayed. Only when all the elaboration errors have been fixed would DesignChecker be able to run the configured synthesis rules. Please note that rules other than synthesis rules are not affected by the presence of elaboration errors and hence these rules will run the analysis and display results normally.

The following figure shows elaboration error violations as displayed in the Results tab.



Synthesis Errors

Synthesis errors result when DesignChecker configured to run some synthesis rules encounters non-synthesizable constructs during design analysis. Some common non-synthesizable constructs are:

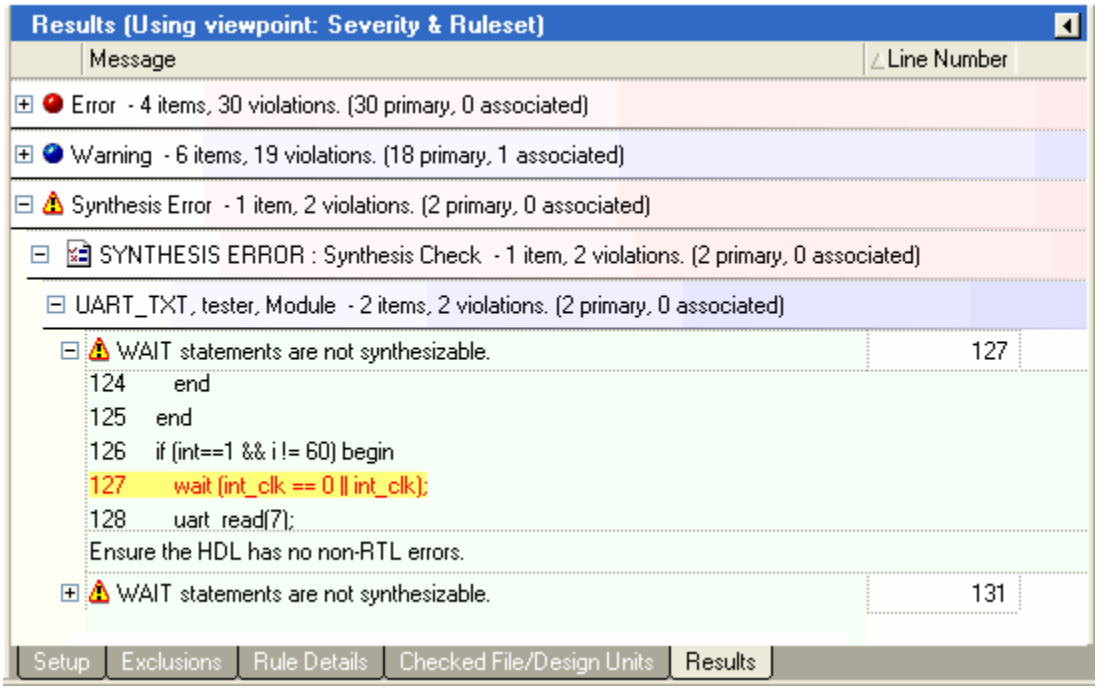
- TIME/REAL/EVENT types
- Classes/Program Blocks/Nested Modules
- Non-unrollable loops
- Mixture of blocking-non blocking assignments on a variable in a single always block (Verilog)
- Non-synthesizable clocking styles (described later in the section)

In such cases, the synthesis rules do not run on the module containing the non-synthesizable constructs. However, the synthesis rules would run on the rest of the modules. Furthermore, rules other than synthesis rules are not affected by non-synthesizable modules and these would run the analysis normally on the non-synthesizable modules as well.

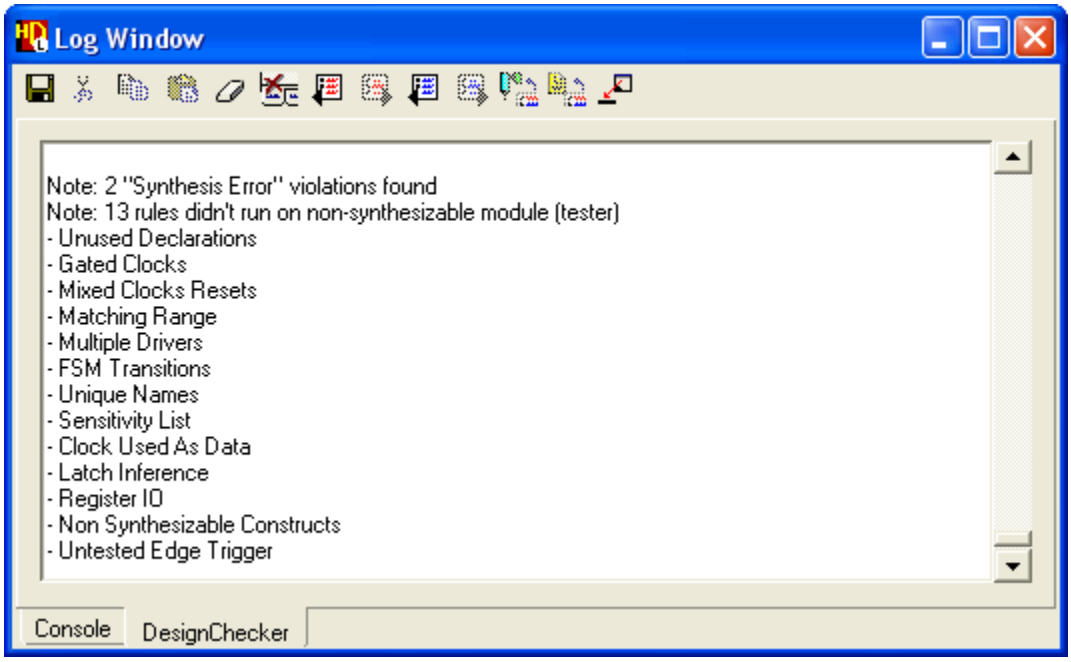
Please note that certain kinds of non-synthesizable constructs in modules are simply ignored by the synthesis rules and the presence of these constructs does not cause DesignChecker to consider the module as non-synthesizable. Some common constructs are:

- Initial blocks
- Delays e.g. #10 in Verilog, AFTER 10 ns in VHDL
- Certain system tasks like $display, $finish etc.
- SV Assertions

The following figure shows synthesis error violations as displayed in the Results tab.



The HDS log window also displays a report of all modules having non-synthesizable constructs along with the rules which did not run on those modules.

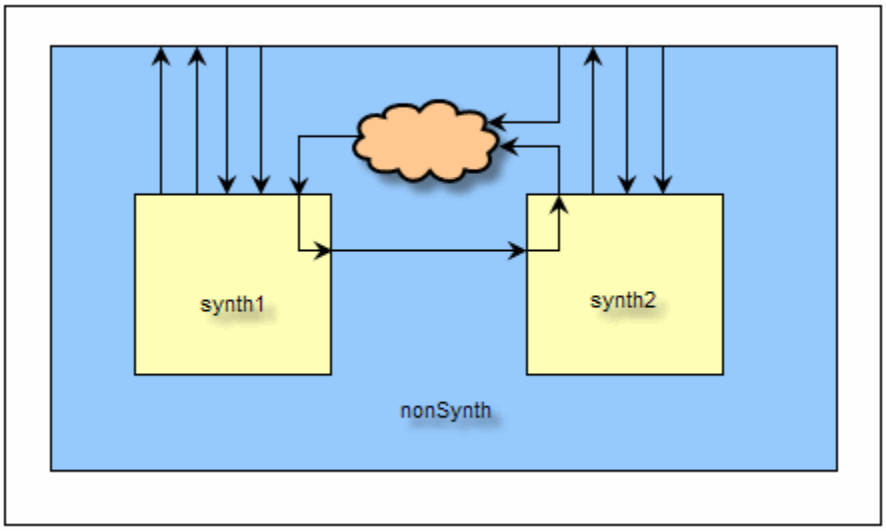


Note

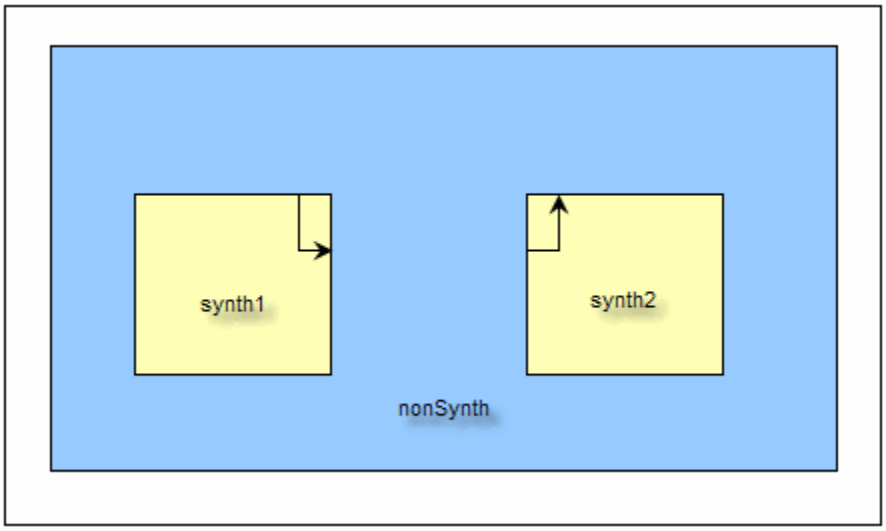
Some non-synthesizable constructs are also offered as dedicated base rules. For example (Conditions – Non-Unrollable Loops or Ranges – Non-Static Ranges). Whenever these rules are part of the active policy, two violations would be produced – one corresponding to the configured rule and a second synthesis error violation.

Synthesizability and implications for design-wide synthesis rules

When a non-synthesizable module is encountered, it is treated as an undefined module and hence logic inside that module is not considered for any of the synthesis rules. However, instances inside that module are considered, but are treated as disconnected from the non-synthesizable module. This means that if the non-synthesizable module contains instances, then the connections of those instances with the parent or sibling instances are not created. Consider the following figure:



In the above figure, a non-synthesizable module, nonSynth instantiates 2 synthesizable modules, synth1 and synth2. Let's assume that the output of synth2 is passed to synth1 through some combinational logic, as depicted by the orange cloud and that there is a combinational feedback loop in the design involving the three modules. In this particular case, DesignChecker synthesis rules view the module nonSynth that contains the non-synthesizable constructs as is shown in the figure below.



Implications of this are:

- If there are some design-wide (inter-module) violations which are confined to the synthesizable sub-hierarchy, synth1/synth2, then these would be reported normally.

- However, if the design-wide violations involve the non-synthesizable module, then they will not be reported. For example, the design actually contains a combinational loop, but since it involves the non-synthesizable module, nonSynth, hence this violation would not be reported.

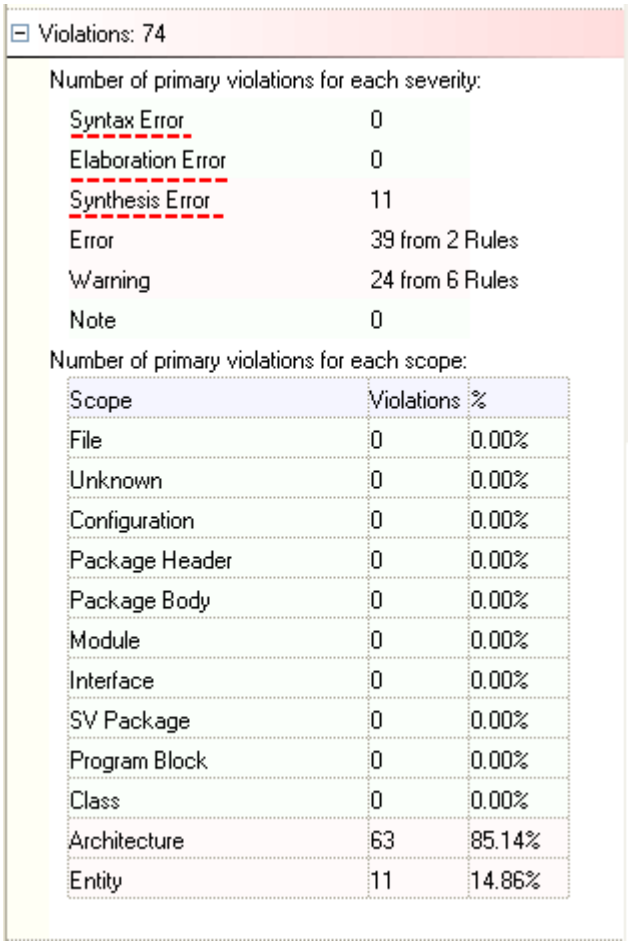
Summary

For DesignChecker synthesis rules to operate properly on the design, it needs to be free of errors. Syntax errors are the most severe errors which prevent any rule checking from running on the design. Elaboration errors preclude any synthesis checks from running on the design, though other rules can run normally. Finally, synthesis errors prevent synthesis rules from running on specific modules, though they may run normally on synthesizable portions of the design. Other checks are not affected by synthesizability of the modules. The following table summarizes the impact of various types of errors on design analysis.

Table B-1. Table 1

Error	Synthesis rules	Other rules
Syntax error	None of the synthesis rules will run on the design	None of the other rules will run on the design
Elaboration error	None of the synthesis rules will run on the design	Will run normally on all modules
Synthesis error	None of the synthesis rules would run on the non-synthesizable modules, however, will run normally on synthesizable modules. Design-wide rules would be affected if they involve the non-synthesizable module.	Will run normally on all modules

The number of Syntax Error, Synthesis Error or Elaboration Error violations found in the design are also reported in the Violations section of the Summary pane as shown in the below figure.



Violations: 74

Number of primary violations for each severity:

Syntax Error	0
Elaboration Error	0
Synthesis Error	11
Error	39 from 2 Rules
Warning	24 from 6 Rules
Note	0

Number of primary violations for each scope:

Scope	Violations	%
File	0	0.00%
Unknown	0	0.00%
Configuration	0	0.00%
Package Header	0	0.00%
Package Body	0	0.00%
Module	0	0.00%
Interface	0	0.00%
SV Package	0	0.00%
Program Block	0	0.00%
Class	0	0.00%
Architecture	63	85.14%
Entity	11	14.86%

Register and Control Signal Inference

Introduction

This section deals with the inference of control and data signals for registers to ease the understanding of violations pertaining to registers and their controls. Several examples are used to give an insight into the mechanism by which control signals are inferred.

Flip-Flop Controls

All registers in the design are mapped to a flip-flop which has a clock, data inputs and some synchronous and asynchronous controls. Please note that only one control of a given type can be inferred for a given flip-flop. The controls which are not active for a particular instance of a flip-flop are effectively grounded, rendering them inactive.

Asynchronous Controls

Asynchronous reset

Asynchronous set

Asynchronous load (+ load data)

Synchronous Controls

Synchronous reset

Synchronous set

Synchronous enable

Please note that a ‘reset’ signal of a flip-flop when asserted will drive a ‘0’ value to the Q output of the flip-flop. A ‘set’ signal is one which when asserted will drive a ‘1’ value to the Q output. Most of the synthesis rules treat both ‘resets’ and ‘sets’ as resets (i.e. a control signal which drives the Q output to a constant).

The asynchronous signals can have any priority set amongst them. However, on the synchronous side, the reset and set should have a higher priority than the enable. If this condition is not satisfied, i.e. the hold condition has a higher priority than the signals which drive the register to a constant, the corresponding signals are not inferred as synchronous set/reset/enable.

The following examples illustrate the corresponding control signals inferred for a given RTL code snippet for a flip-flop.

Example 1

RTL

```
always @(posedge clk or negedge neg_a_rst or posedge pos_a_set or negedge
neg_a_load)
   if (!neg_a_rst)
      q <= 0;
   else if (pos_a_set)
      q <= 1;
   else if (!neg_a_load)
      q <= !load_data;
   else if (pos_s_set)
      q <= 1;
   else if (!neg_s_rst)
      q <= 0;
   else if (pos_s_en)
      q <= !d;
```

Control signals inferred

Active low asynchronous reset: **neg_a_rst**

Active high asynchronous set: **pos_a_set**

Active low asynchronous load: **neg_a_load**

Asynchronous load data: **!load_data**

Active high synchronous set: **pos_s_set**

Active low synchronous reset: **neg_s_rst**

Synchronous clock enable: **pos_s_en**

Example 2

RTL

```
always @(posedge clk or negedge neg_a_set or posedge pos_a_load)
   if (!neg_a_set)
      q <= 1;
   else if (pos_a_load)
      q <= load_data;
   else if (pos_s_rst)
      q <= 0;
   else
      q <= d;
```

Control signals inferred

Active low asynchronous set: **neg_a_set**

Active high asynchronous load: **pos_a_load**

Asynchronous load data: **load_data**

Active high synchronous reset: **pos_s_rst**

Asynchronous control signals

As already mentioned earlier, the asynchronous control signals have the highest priority. In addition, a flip-flop can have only one of a particular type of asynchronous control input. Finally, the asynchronous control signals are allowed to have any priority set among them another. The following examples demonstrate the asynchronous control signals inferred for different cases.

Example 3

RTL

```
always @(posedge clk or posedge rst1 or negedge rst2)
   if (rst1)
```

```
      q <= 0;
   else if (!rst2)
      q <= 0;
   else
      q <= d;
```

Control signals inferred

Active high asynchronous reset: **~((~rst1) & rst2)**

Note

Since only one asynchronous reset can be inferred for a flip-flop, hence a logic function involving rst1 and rst2 forms the asynchronous reset of the flip-flop. Please note that this is considered as an internally generated reset. rst1 and rst2 will not be considered as asynchronous resets because neither of these signals are resets in their own right.

Example 4

RTL

```
always @(posedge clk or posedge ld or negedge rst2 or posedge set)
   if (ld)
      q <= ldata;
   else if (!rst2)
      q <= 0;
   else if (set)
      q <= 1;
   else
      q <= d;
```

Control signals inferred

Active high asynchronous set: **set**

Active low asynchronous reset: **rst2**

Active high asynchronous load: **ld**

Asynchronous load data: **ldata**

Note

Asynchronous control signals are allowed to have any priority set among them. In the above case, the flip-flop inferred will have the asynchronous load that has the highest priority signal.

Example 5

RTL

```
always @(posedge clk or posedge rst2 or negedge ld or negedge rst1 or
posedge set)
   if (set)
      q <= 1;
   else if (!rst1)
      q <= 0;
   else if (!ld)
      q <= ldata;
   else if (rst2)
      q <= 0;
   else
      q <= d;
```

Control signals inferred

Active high asynchronous set: **((ldata & (rst1 & (~ld))) | set)**

Active high asynchronous reset: **((~set) & (combinational function of rst1, ld, ldata, rst2))**

Note

In the above example, ‘q’ is asynchronously driven to a value of zero under the influence of more than one signal and distinct asynchronous control signals cannot be inferred. In such cases, the tool will map the entire asynchronous control logic to the asynchronous set and reset inputs of the flip-flops.

Synchronous control signals

As mentioned earlier, the tool infers synchronous set, synchronous reset and synchronous enable for flip-flops. No more than one of each type of synchronous control input can be present for a flip-flop. The synchronous set or reset can have any priority set between them. The synchronous enable always has a lower priority than synchronous set and reset.

Synchronous control signals inference could be trickier than asynchronous control signals inference because synchronous set and reset signals can always be made a part of the logic feeding the D input of a flip-flop. Conversely, a complex logic feeding the D input of a flip can often be broken down to extract a synchronous set and/or a synchronous reset. Consider the following example:

Example 6

```
always @(posedge clk)
   if (!rst)
      q <= 0;
   else if (set)
      q <= 1;
   else
      q <= d;
```

The above RTL can be modeled as a flip-flop with the following synchronous control signals and D input:

Active low synchronous reset: **rst**

Active high synchronous set: **set**

D input: **d**

However the following two alternate inferences are also possible:

Alternate Inference 1

Active low synchronous reset: **rst**

D input: **(set | d)**

Alternate Inference 2

D input: **(rst & (set | d))**

Further, consider the following example:

Example 7

```
always @(posedge clk)
      q <= rst & d;
```

In the above example, either the D-input logic can be inferred to be ‘rst & d’, in which case no synchronous reset signal is inferred, or, an active low synchronous reset can be inferred either as ‘rst’, in which case the D input is fed by ‘d’ alone.

Hence, in order to be unambiguous and intuitive as far as possible, the tool follows some rules for synchronous control detection.

Rule 1 – If – else if structure

Synchronous set and reset signals are usually coded as part of a control flow that precedes the D input assignment (like the if-else-if control flow in Example 6). Hence, the tool will infer a synchronous set and a synchronous reset for Example 6, but not for Example 7.

Rule 2 – Single RTL Signal

Since control flows can be also be part of combinational logic other than synchronous sets or resets, the tool avoids inferring synchronous sets or resets for any arbitrary logic that eventually results in driving a ‘1’ or ‘0’ to the Q output of a flip-flop. This is achieved by searching for a single RTL signal in the control flow which results in a ‘1’ or ‘0’ assignment to the Q output. If such a signal is found, a corresponding synchronous set or reset signal will be inferred, or else, the signals are made part of the D input logic.

Rule 3 – Non-constant D-input

A non-constant D input logic must also be inferred to distinguish the user intent of a synchronous control being inferred for a flip-flop. This is because the synchronous reset and set are supposed to always drive a constant ‘0’ or a ‘1’ to the Q output, a characteristic that is used as a distinguishing factor for treating them differently from the D input logic.

The following examples show the synchronous control and D input inference for various scenarios.

Example 8

RTL

```
always @(posedge clk)
   if (set)
      q <= 1;
   else if (rst)
      q <= 0;
   else if (en)
      q <= d;
```

Control signals and D input inferred

Active high synchronous set: **set**

Active high synchronous reset: **rst**

Synchronous enable: **en**

D input: **d**

Example 9

RTL

```
always @(posedge clk)
   if (en)
      q <= d;
   else if (set)
      q <= 1;
   else if (rst)
      q <= 0;
```

Control signals and D input inferred

Synchronous enable: ((rst | set) | en)

D input: (combinational function of d, set, en)

Note

Here no synchronous set or reset was inferred because a synchronous set or reset can be inferred only if it is the highest priority signal (over any other logic assignment to the Q output through the D input).

Example 10

RTL

```
always @(posedge clk)
   if (rst)
      q <= 0;
   else if (en)
      q <= d;
   else if (set)
      q <= 1;
```

Control signals and D input inferred

Active high synchronous reset: **rst**

Synchronous enable: **(en | set)**

D input: **(d | (~en))**

Example 11

RTL

```
always @(posedge clk)
   if (rst)
      q <= 0;
   else if (!set)
      q <= 1;
   else if (en)
      q <= 1;
   else
      q <= 0;
```

Control signals and D input inferred

Active high synchronous reset: **rst**

Active low synchronous set: **set**

D input: **en**

Example 12

RTL

```
always @(posedge clk)
if (rst)
q <= 0;
else if (!set)
q <= 1;
else if (en)
q <= 1;
```

Control signals and D input inferred

Synchronous enable: **~(((~en) & set) & (~rst))**

D input: **~rst**

Note

Here the tool fails to infer synchronous set or reset because a synchronous enable logic has to be inferred (to model the Q output retaining behavior) and there is no other explicit D input logic specified in the RTL, in which case '~rst' itself becomes the D input logic.

Example 13

RTL

```
always @(posedge clk)
if (rst)
q <= 0;
else if (!rst2)
q <= 0;
else if (en)
q <= d;
```

Control signals and D input inferred

Active high synchronous reset: **rst**

Synchronous enable: **~((~en) & rst2)**

D input: **(d & rst2)**

Example 14

RTL

```
always @(posedge clk)
if (rst & !rst2 || en)
q <= 0;
```

```
else
q <= d;
```

Control signals and D input inferred

D input: **(d & ~((rst & (~rst2)) | en))**

Note

As mentioned earlier, in the absence of a single RTL signal that can be inferred as a synchronous set or reset, the tool will infer the complete logic on the D-side of the flip-flop. This is because the tool could otherwise infer arbitrary synchronous sets or resets from normal combinational logic anywhere in a design, resulting in confusing, albeit functionally correct control logic for flip-flops and subsequent misleading violations for reset related rules.

Counter controls inference

The tool also infers synchronous counters. The inferred synchronous counters have a similar but different set of control signals than the flip-flops. Since the number of counters is generally much less than the number of flip-flops in a design, synchronous counter inference is typically aggressive in nature. The counter inference process is based on a pre-defined template that the tool adheres to while inferring counter control and data signals.

The only possible control signals for a counter are

Asynchronous side

Asynchronous reset

Asynchronous set

Synchronous side

Clock enable

Synchronous reset

Synchronous set

Synchronous load (+ load data)

Count enable

Up-down control

The general model of a counter is given below.

```
always@(posedge clock or posedge aclear or posedge aset)
   if (aclear)
```

```
   q <= {WIDTH{1'b0}};
else if (aset)
   q <= {WIDTH{1'b1}};
else if (clk_en)
   if (sclear)
   q <= {WIDTH{1'b0}};
   else if (sset)
   q <= {WIDTH{1'b1}};
   else if (sload)
   q <= data;
   else if (cnt_en)
   if (updn)
   q <= q + 1;
   else
   q <= q - 1;
```

The control signals in the above model are listed as follows

Asynchronous reset: **aclear**

Asynchronous set: **aset**

Clock enable: **clk_en**

Synchronous reset: **sclear**

Synchronous set: **sset**

Synchronous load: **sload**

Count enable: **cnt_en**

Up-down control: **updn**

Please note that as with flip-flops, the asynchronous set and reset signals can have any priority set among them. Similarly, the synchronous set and reset signals can have any priority set among them. The remaining control signals will have their relative priorities exactly as shown in the counter model. If a given RTL cannot be mapped to a valid synchronous counter model, a flip-flop with appropriate control and data logic for each assigned bit will be inferred instead.

The counter control signals inferred under various scenarios are shown in the examples that follow.

Example 15

RTL

```
always @(posedge clk or posedge rst)
if (rst)
   q <= 0;
else if (clk_en & en2)
   if (s_rst & s_rst2)
      q <= 0;
```

```
else if (cnt_en)
   q <= q + 1;
```

Control signals inferred

Active high asynchronous reset: **rst**

Clock enable: **(clk_en & en2)**

Active high synchronous reset: **(s_rst & s_rst2)**

Count enable: **cnt_en**

Example 16

RTL

```
always @(posedge clk or posedge rst or posedge set)
   if (set)
      q <= {5{1'b1}};
   else if (rst)
      q <= 0;
   else if (s_rst & s_rst2)
      q <= 0;
   else if (cnt_en)
      q <= q + 1;
```

Control signals inferred

Active high asynchronous reset: **rst**

Active high asynchronous set: **set**

Active high synchronous reset: **(s_rst & s_rst2)**

Count enable: **cnt_en**

Behavior with constant/unused flip-flops or counters

If a flip-flop drives a fixed, constant value under all conditions, then a stuck-at-0 or stuck-at-1 violation will appear for the flip-flop accordingly (if the corresponding rule is enabled) and the flip-flop will be optimized. For instance, the constant optimized flip-flops will produce no reset related violations as shown in the example below.

Example 17

RTL

```
module dut(input clk, rst, en, input [5:0] d_in, output [4:0] q_out);
wire [5:0] d;
```

```
reg [5:0] q;

assign q_out = q[4:0];
assign d[5:1] = d_in[5:1];
assign d[0] = 0;
always @(posedge clk)
   if (rst)
      q <= 0;
   else if (en)
      q <= d;
endmodule
```

Example violations generated

State bit(s) ‘q[0]’ has/have a stuck-at-0 fault.

Net ‘d_in[0]’ is unused.

Net ‘q[5]’ is unused.

Register ‘q[5:1]’ has not been initialized using asynchronous reset.

As seen above, although flip-flop ‘q[5]’ is unused, it still appears in a reset related violation. However, the flip-flop ‘q[0]’ is optimized to a constant ‘0’ and no other violations apart from a stuck-at-0 violation appears for it.

‘Z’ Propagation Across Flip-Flops

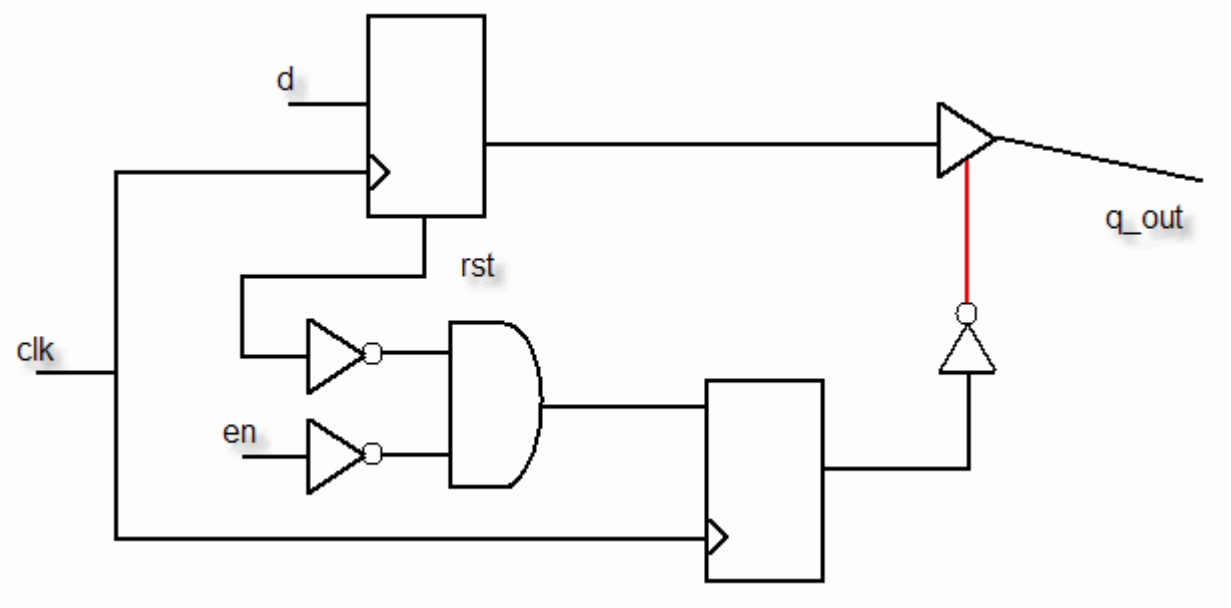
Because of the way flip-flops are designed, the input of a flip-flop can never assume a value ‘z’. So if the RTL source code is such that a ‘z’ needs to be propagated across a flip-flop then some additional logic needs to be created. This ensures that synthesis and simulation results match. Please refer to the example below.

Example 18

RTL

```
module dut(input clk, rst, en, input d, output q_out);
always @(posedge clk)
if (rst)
q_out <= 1’b0;
else if (en)
q_out <= d;
else
q_out <= 1’bz;
endmodule
```

In this case, the synthesized logic will look like



As seen from the figure above, the ‘Z’ is completely removed from the input cone of the flip-flop. Whenever rst or en goes high, the enable of the tristate buffer is enabled and hence it acts as a buffer transmitting the input (0 if rst is high, or d if en is high) to the q_out. However, if rst and en both are low, then the tristate is disabled and q_out is consequently tristated. Thereby the simulation and synthesis behavior is kept consistent.

Please note that the tristate enable shown as the bold red line in the figure, is not really an RTL signal, rather an internally generated one. Hence, this signal will not get reported as the tristate enable.

Finite State Machines (FSMs)

Introduction

In order to help understand the subtle nature of some of the FSM related violations and reports generated by DesignChecker, the following sections provide useful reference information. The information tackles the internal mechanisms of the tool describing how an FSM is inferred, what states are reported or unreported and some of the violations that are generated.

Necessary Conditions for FSM inference

Some conditions are necessary for DesignChecker to be able to infer an FSM. These conditions are explained in detail below

State variable inference

The FSM extraction process starts from the deduction of a potential state variable. Such a state variable would typically be inferred from the expressions of case statements or equivalent if-else-if statements written in the RTL. For example, in Code Snippet 1 given below, the FSM extraction method will assume ***curr_state*** as a possible state variable (because of its presence in a case expression) and start the FSM extraction process. If no such case or equivalent if statement is found, an FSM will not be extracted. The assignments to the already inferred state variables will result in further state variables being inferred. In Code Snippet 1, the other state variable inferred for the same FSM is ***next_state***.

Constant reads and writes of state variables

Each of the state variables inferred in the course of the FSM inference process needs to necessarily have assignments to or comparisons with constant values only, with the exception of assignments to other state variables. Other non-constant assignments/comparisons of state variables will invalidate the FSM. In Code Snippet 1, both ***curr_state*** and ***next_state*** are assumed to be state variables and the FSM extraction succeeds because all assignments and comparisons of both ***curr_state*** and ***next_state*** are with constant values.

Note

All state variables inferred in the FSM extraction process must be assigned at least once with either a constant value or another state variable.

State dependent transition

Mere assignments of state variables to constant values are not sufficient for an FSM extraction to succeed. DesignChecker must also be able to infer a state transition from one state to another for a valid FSM to get extracted. For example, in Code Snippet 1, the tool sees an assignment to ***next_state*** that depends on ***curr_state***. On the other hand, consider the example in Code Snippet 2 where all the reads and writes of the inferred state variable, ***curr_state*** are with constant values. However, there is no state assignment that is dependent on the value of ***curr_state*** and all the states are assigned independently. In such a case, the tool will not infer the logic involving ***curr_state*** as an FSM.

Presence of a clocked transition

Transitions of states on the edge of a clock are necessary for a valid FSM to get extracted. If no clocked transitions are found during the FSM extraction process, the corresponding FSM will be invalidated. In Code Snippet 1, DesignChecker sees the assignment of ***curr_state*** to ***next_state*** in a clocked always block. If such a clocked block was not found, the FSM would not have been extracted.

```
module FSM(input clock, reset, in1, in2, output reg out1, out2, out3);
   reg [2:0] curr_state, next_state;
   parameter s0 = 0, s1 = 1, s2 = 2, s3 = 3;
always @(posedge clock or negedge reset) begin
      if (!reset)
         curr_state <= s0;
      else
         curr_state <= next_state;
   end
always @* begin
      case (curr_state)
         s0: if (in1) begin
                next_state <= s1;
             end
         s1: if (in1 || in2) begin
                next_state <= s3;
             end
         s2: if (in1) begin
                next_state <= s2;
             end
             else begin
                next_state <= s3;
             end
         s3: if (in2) begin
                next_state <= s0;
             end
         default ;
      endcase
   end
always @* begin
      case (curr_state)
         s0: if (in1)
                out1 <= 1;
         s1:    out2 <= 1;
         s2: if (in2)
                out3 <= 1;
      endcase
   end
endmodule
```

Code Snippet 1

```
module NO_FSM(input in1, in2, clk, rst, output reg out1, out2);
   reg [1:0] curr_state;
always @*
      case (curr_state)
         2'b00: out1 <= 1;
         2'b11: out2 <= 0;
         default: out1 <= 0;
      endcase
always @(posedge clk)
      if (rst)
         curr_state <= 2'b00;
      else if (in1)
         curr_state <= 2'b01;
      else if (in2 || in1)
         curr_state <= 2'b11;
      else
         curr_state <= 2'b01;
endmodule
```

Code Snippet 2

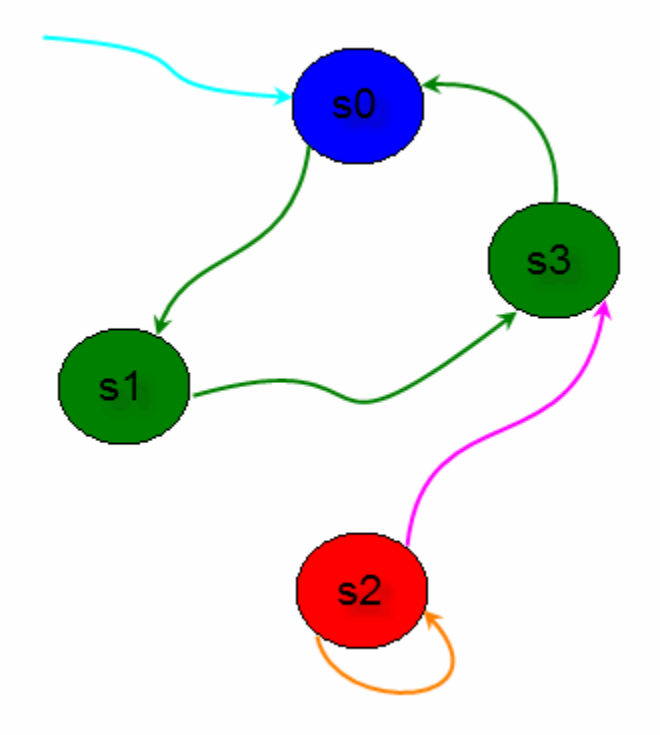
FSM-related violations

Some of the important FSM-related rules deal with reachability of states and default detection. This subsection describes how these aspects of FSMs are checked by DesignChecker.

Externally reachable states and unreachable states

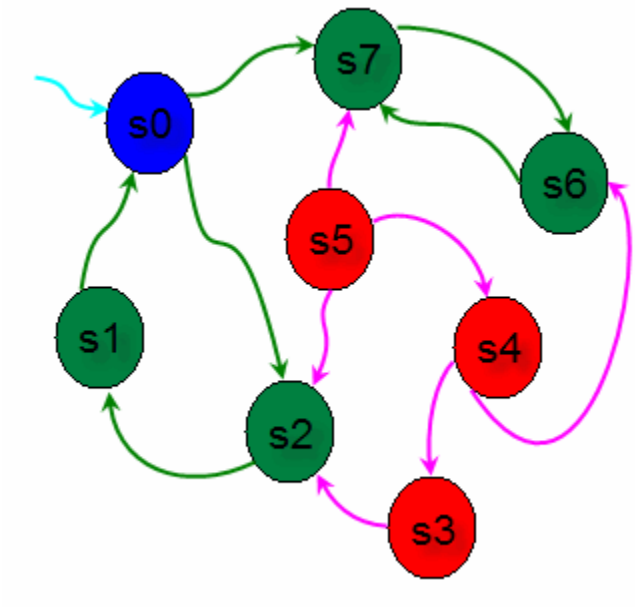
DesignChecker performs reachability analysis of states starting from a set of ***externally reachable states***. An externally reachable state in an FSM is a state to which the FSM can reach independent of any of the states of the FSMs. If no externally reachable states are found during the FSM extraction process, but all other valid conditions for FSM extraction are met, an FSM with no externally reachable states will be reported by the tool. In Code Snippet 1 given above, the assignment of ***s0*** to ***curr_state*** is independent of any of the states of the FSM and hence the state ***s0*** is an externally reachable state. The reachability analysis will hence start from ***s0*** and the set of reachable states deduced is ***{s0, s1, s3}***. Note that the state ***s2*** is unreachable and hence, the FSM report will not show the state ***s2*** as a part of the inferred FSM. This situation has been diagrammatically presented in Figure 1. Also, the tool will report the state ***s2*** as having no incoming transitions. Please note that a transition from a state to itself (orange arrow in Figure 1) is not considered as either an incoming or an outgoing transition.

Figure B-1. Figure 1



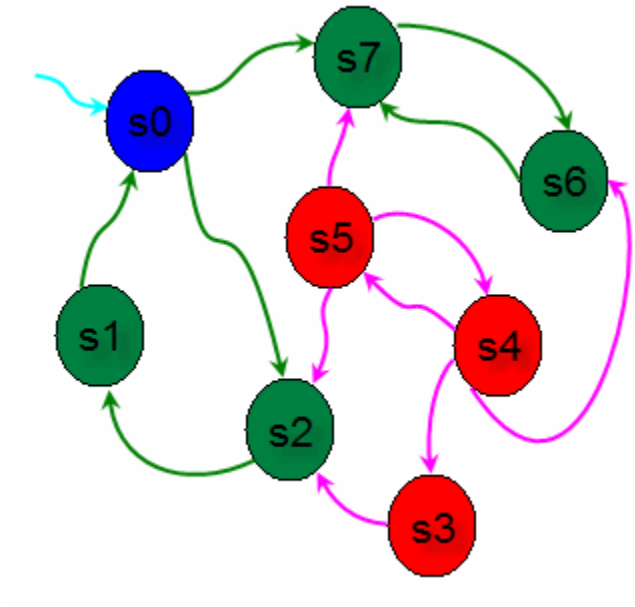
As a further example, consider an FSM with a corresponding transition graph as shown in Figure 2. In this FSM, the externally reachable state is ***s0***. Performing a reachability analysis from ***s0***, we arrive at the set of reachable states as ***{s0, s1, s2, s6, s7}*** and the same will be seen in the FSM report. The set of unreachable states of this FSM is ***{s3, s4, s5}***. Since ***s5*** is a state with no incoming transitions, the tool will report a no-incoming transition primary violation for ***s5***. The states ***s3*** and ***s4*** are unreachable because ***s5*** has no incoming transitions and they are reachable only from ***s5***. Hence the tool will also report unreachable state violations for ***s3*** and ***s4*** as associated violations of the no-incoming transition (primary) violation for ***s5***.

Figure B-2. Figure 2



Further, consider an FSM with the transition graph as in Figure 3. In this case, the set of states ***{s3, s4, s5}*** are still unreachable. However, all the unreachable states have incoming transitions (albeit from unreachable states). In this case, 3 independent “Unreachable state” violations would be populated.

Figure B-3. Figure 3



Default handling detection

The purpose of default handling check in FSMs is to detect that the FSM either never goes to an invalid state or, if it is possible that the FSM may go into an invalid state, then it is able to recover by going to a valid state in the subsequent cycle. Hence for this check, DesignChecker tries to figure out the presence of a default branch and an assignment to a state variable in the default branch (if all possible choices have already not been covered in the case statement). If such a default handling is not found, the tool will populate a default handling violation. In Code Snippet 1 above, even though there is a default branch, the tool will report a default handling violation because there is no state assignment in the default branch, and all possible choices are already not covered because the state variable is of three bits and can take eight possible values (but only four choices are covered).

Miscellaneous

Enum encoding in VHDL

Enumerated types in VHDL do not have an encoding associated with them by default. DesignChecker encodes such enums using binary encoding. For example, if an enum type is declared as:

```
type myenum is (S0, S1, S2, S3, S4);
signal x : myenum;
```

then the enum literals are encoded as follows:

S0 = “000”

S1 = “001”

S2 = “010”

S3 = “011”

S4 = “100”

Since there are 5 possible values, these can be represented in a minimum of 3 bits and hence the signal *x* is assumed to be 3-bit wide.

Reset/Preset detection for enum type registers

Consider the following snippet of code:

```
type myenum is (S0, S1, S2, S3, S4);
signal x,y : myenum;

process (clk, areset)
begin
```

```
    if(areset = '1') then
        x <= S1;
    elsif (clk'event and clk = '1') then
        x <= y;
    end if;
end process;
```

Assuming the encoding as mentioned above, the code would be interpreted as:

```
if(areset = '1') then
        x(2 downto 0) <= "001";
```

This means that areset serves as an asynchronous reset for *x(2 downto 1)* and an asynchronous preset for *x(0)*. This can lead to unintuitive violations for reset-related design-wide rules like "Mixed Clocks-Resets" and "Reset Logic Function". This situation will typically occur whenever the reset value is not the first enum literal. (i.e. all zeros)

Ignored Code

Synthesis rules do not consider unreachable code while performing design checking. Unreachable code includes the following:

- Blocks that are unreachable due to impossible conditions governing their activation. Such code is flagged by the "Unreachable Blocks" rule.
- Code under synthesis pragmas like "translate_off/on" and "synthesis_off/on".
- Initial blocks

Unreachable code in the design may lead to seemingly incorrect violations. Consider the following snippet of code:

```
input in1;

always @(in1)
begin
    if (in1 == 3'b000)
        done = 1'b1;
    else if (in1 == 3'b101)
        waiting = 1'b1;
    else
        busy = 1'b1;
```

Since in1 is single-bit wide, hence the possible values it can take are 0 and 1, which have been already covered by the first two conditions. As a result, the final "else" is unreachable and hence even though it appears from the source code, that signal "busy" is driven, DesignChecker will not see it as driven.

Tristates and Enables

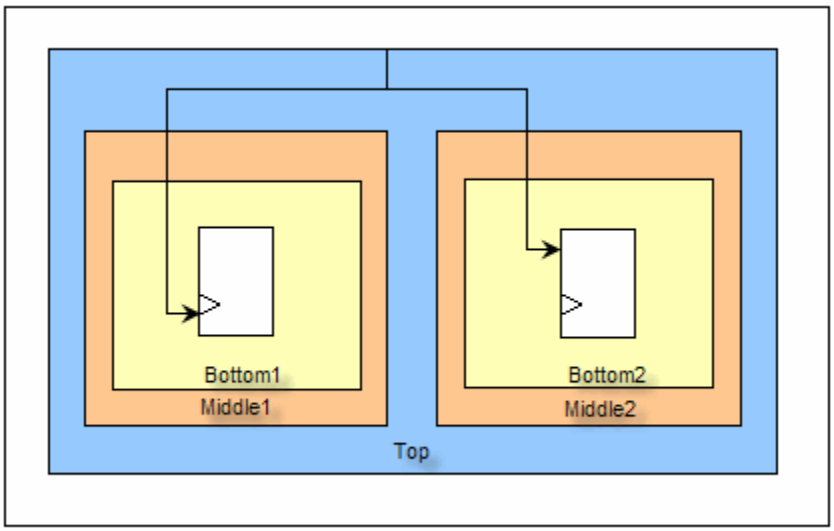
Tristate buffers have an enable signal, which when asserted, causes it to act as a buffer. Combinatorial logic which enables the tristate buffer will not be considered for reporting purposes. For example

```
always @(sel1, sel2, in1)
begin
    if (sel1 || sel2)
        out1 = in1;
    else
       out1 = 1'bz;
```

In the above case, sel1/sel2 will not be detected as tristate enables, since neither of these are enables in their own right.

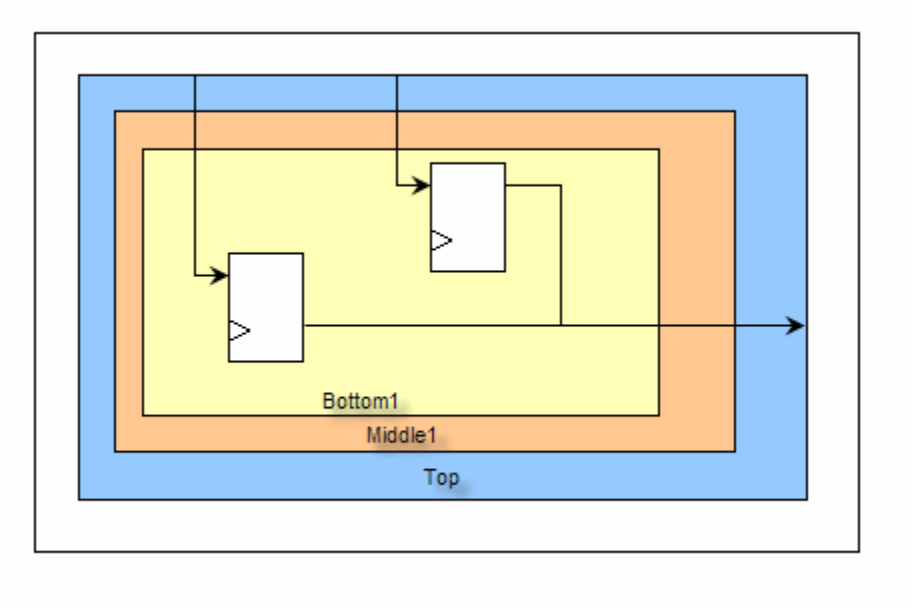
Design-wide (Hierarchical) Rules

A design-wide check from a synthesis rule perspective is one which can extend across design units and hence depends on the design context. Hence the violations for design-wide rules produced in a design unit may change depending upon the selection of the design root. Consider the following design hierarchy.



As seen in the above figure, if “top” is selected as the design root, then the “Clock Used as Data” violation will be reported. However, if “Middle1” or “Bottom1” or any other design unit is selected as the design root, this violation will not be detected. Hence this is a design-wide check.

On the other hand, consider the following design hierarchy:



The figure above represents a multiple driver scenario wherein a net is driven by 2 registers. In this case, the “Multiple Drivers” violation will be reported independent of whether “top” or “Middle1” or “Bottom1” is selected as the design top. In other words, this rule is independent of the design context and consequently is not a design-wide check.

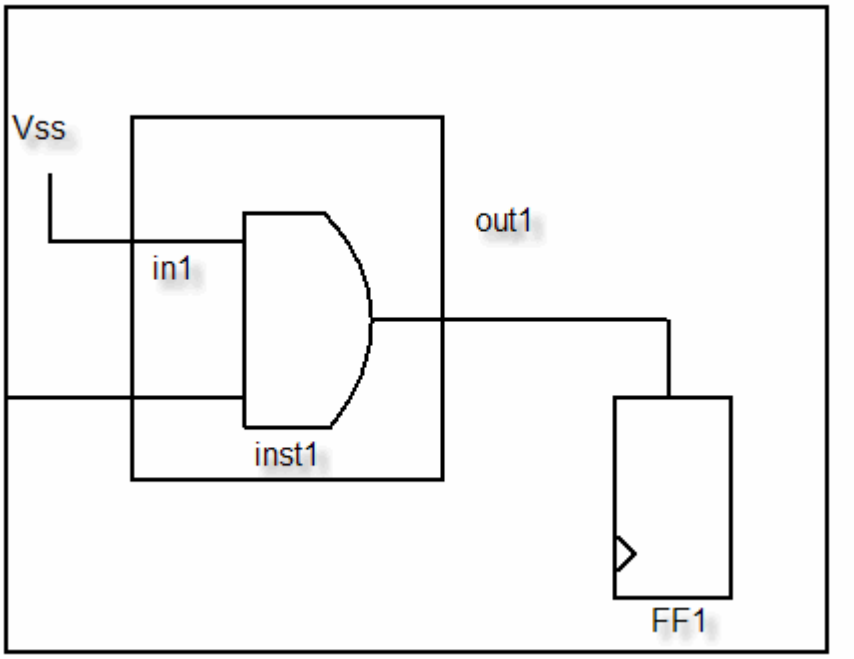
Unused Combinational/Sequential Logic and design-wide rules

Design units may have unused combinational logic or unused sequential logic (registers/latches which do not impact the primary output of the design). DesignChecker optimizes any unused combinational logic while performing design-wide rules. However, unused sequential logic is preserved by DesignChecker. The reason for this is that typically sequential logic forms “preserve-points” for the design and hence DesignChecker does not optimize them even if they do not contribute to the primary outputs of the design. As a result, design-wide rules which deal with combinational logic, like “Combinational Feedback Loops” will not report violations if the combinational logic involved in the violation is unused. On the other hand, design-wide rules which deal with sequential logic, like “Internally Generated Clocks” would report violations even if the sequential element involved in the violation is unused.

Constant Propagation across design hierarchy for design-wide rules

For DesignChecker, module boundaries are sacrosanct, and no optimizations or constant propagation is done across module boundaries. Hence if some logic inside a module can be

optimized based on the inputs connected to an instance of that module, such logic will not be optimized. Please consider the figure below



As can be seen from the figure above, since the input in1 of the instance inst1 is grounded, hence the output out1 will always be 0, which in turn means that the reset of the register FF1 is tied low. However, since DesignChecker does not perform any optimizations across module boundaries, hence it will not be able to determine that the output out1 will be tied low, and hence the fact that FF1 is tied low will not be detected. Please note that if the complete logic were in a single module, then this would be detected.

Index

— S —

— T —

— V —

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. **ORDERS, FEES AND PAYMENT.**

   1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement ("Order(s)"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.

   1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.

   1.3. All Products are delivered FCA factory (Incoterms 2000), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer requests any change or enhancement to Software, whether in the course of

receiving support or consulting services, evaluating Software, performing beta testing or otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

4. **BETA CODE.**

4.1. Portions or all of certain Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.

4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. **RESTRICTIONS ON USE.**

5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use it except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Software or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Software, or disclose to any third party the results of, or information pertaining to, any benchmark.

5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.

5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms

of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

5.4. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at http://supportnet.mentor.com/about/legal/.

7. **AUTOMATIC CHECK FOR UPDATES; PRIVACY.** Technological measures in Software may communicate with servers of Mentor Graphics or its contractors for the purpose of checking for and notifying the user of updates and to ensure that the Software in use is licensed in compliance with this Agreement. Mentor Graphics will not collect any personally identifiable data in this process and will not disclose any data collected to any third party without the prior written consent of Customer, except to Mentor Graphics' outside attorneys or as may be required by a court of competent jurisdiction.

8. **LIMITED WARRANTY.**

8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

9. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **HAZARDOUS APPLICATIONS.** CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY ("HAZARDOUS APPLICATIONS"). NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10. THE PROVISIONS OF THIS SECTION 11 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

12. **INFRINGEMENT.**

12.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance

to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

12.2. If a claim is made under Subsection 12.1 Mentor Graphics may, at its option and expense, (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.

12.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.

12.4. THIS SECTION 12 IS SUBJECT TO SECTION 9 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS FOR DEFENSE, SETTLEMENT AND DAMAGES, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.

13. **TERMINATION AND EFFECT OF TERMINATION.** If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term.

13.1. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.

13.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.

14. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products and information about the products to certain countries and certain persons. Customer agrees that it will not export Products in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.

15. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.

16. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.

17. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FLEXlm or FLEXnet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 17 shall survive the termination of this Agreement.

18. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the United States. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, USA, if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International

Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. This section shall not restrict Mentor Graphics' right to bring an action against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

19. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

20. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing by authorized representatives of the parties. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 100615, Part No. 246066