cādence®

Virtuoso Hierarchy Editor User Guide

Product Version ICADVM20.1
October 2020

Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor

Contents

Preface

The *Virtuoso Hierarchy Editor User Guide* describes the Virtuoso® Hierarchy Editor, which lets you browse a design hierarchy and create and edit design configurations.

This user guide is aimed at developers and designers of integrated circuits and assumes that you are familiar with:

- The Virtuoso design environment and application infrastructure mechanisms designed to support consistent operations between all Cadence® tools.
- The applications used to design and develop integrated circuits in the Virtuoso design environment, notably, the Virtuoso Layout Suite, and Virtuoso Schematic Editor.
- The Virtuoso design environment technology file.

This preface contains the following topics:

- Scope
- Licensing Requirements
- Related Documentation
- Additional Learning Resources
- Customer Support
- Feedback about Documentation
- Typographic and Syntax Conventions

Scope

Unless otherwise noted, the functionality described in this guide can be used in both mature node (for example, IC6.1.8) and advanced node and methodologies (for example, ICADVM20.1) releases.

Label	Meaning
(ICADVM20.1 Only)	Features supported only in ICADVM20.1 advanced nodes and advanced methodologies releases.
(IC6.1.8 Only)	Features supported only in mature node releases.

Licensing Requirements

To run Pin Checker in the Virtuoso Hierarchy Editor window, Virtuoso searches for one of the licenses mentioned below in the following order:

- License 21400: Virtuoso Schematic Editor Verilog Interface
- License 21060: Virtuoso Schematic VHDL Interface
- License 206: Virtuoso Simulation Environment

For information on licensing in the Virtuoso design environment, see the *Virtuoso Software Licensing and Configuration Guide*.

Related Documentation

What's New and KPNS

Virtuoso Hierarchy Editor What's New

Virtuoso Hierarchy Editor Known Problems and Solutions

Installation, Environment, and Infrastructure

- *Virtuoso Schematic Editor User Guide*

- *Virtuoso Unified Custom Constraints User Guide*
- *Cadence Application Infrastructure User Guide*

Additional Learning Resources

Video Library

The Video Library on the Cadence Online Support website provides a comprehensive list of videos on various Cadence products.

To view a list of videos related to a specific product, you can use the *Filter Results* feature available in the pane on the left. For example, click the *Virtuoso Layout Suite* product link to view a list of videos available for the product.

You can also save your product preferences in the Product Selection form, which opens when you click the *Edit* icon located next to *My Products*.

Virtuoso Videos Book

You can access certain videos directly from Cadence Help. To learn more about this feature and to access the list of available videos, see Virtuoso Videos.

Rapid Adoption Kits

Cadence provides a number of Rapid Adoption Kits that demonstrate how to use Virtuoso applications in your design flows. These kits contain design databases and instructions on how to run the design flow.

To explore the full range of training courses provided by Cadence in your region, visit Cadence Training or write to training_enroll@cadence.com.

Note: The links in this section open in a separate web browser window when clicked in Cadence Help.

Help and Support Facilities

Virtuoso offers several built-in features to let you access help and support directly from the software.

- The Virtuoso *Help* menu provides consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.
- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

  The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the *Home* button on the Cadence Help toolbar (provided you have not set a custom home page).

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

  Cadence is committed to keeping your design teams productive by providing answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit https://www.cadence.com/support.
- Log on to Cadence Online Support

  Customers with a maintenance contract with Cadence can obtain the latest information about various tools at https://support.cadence.com.

Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.

- On the Cadence Online Support Product Manuals page, select the required product and submit your feedback by using the *Provide Feedback* box.

Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

text	Indicates names of manuals, menu commands, buttons, and fields.
`text`	Indicates text that you must type is same as presented. Typically used to denote command, function, routine, or argument names that must be typed literally.
`z_argument`	Indicates text that you must replace with an appropriate argument value. The prefix (in this example, *`z_`*) indicates the data type the argument can accept and must not be typed.
|	Separates a choice of options.
`{ }`	Encloses a list of choices, separated by vertical bars, from which you **must** choose one.
`[ ]`	Encloses an optional argument or a list of choices separated by vertical bars, from which you **may** choose one.
`[ ?argName` *`t_arg`* `]`	
	Denotes a key argument. The question mark and argument name must be typed as they appear in the syntax and must be followed by the required value for that argument.
`...`	Indicates that you can repeat the previous argument. Used with brackets to indicate that you can specify zero or more arguments. Used without brackets to indicate that you must specify at least one argument.
`,...`	Indicates that multiple arguments must be separated by commas.
`=>`	Indicates the values returned by a Cadence® SKILL® language function.
`/`	Separates the values that can be returned by a Cadence SKILL language function.

If a command-line or SKILL expression is too long to fit within the paragraph margins of this document, the remainder of the expression is moved to the next line and indented. In code excerpts, a backslash (\ ) indicates that the current line continues on to the next line.

1

Virtuoso Hierarchy Editor Overview

This chapter covers the following topics:

- Viewing the Legend on page 74
- Changing the Hierarchy Editor Fonts on page 76
- Saving Custom Settings on page 78
- Exiting the Hierarchy Editor on page 80

What You Can Do with the Hierarchy Editor

The Virtuoso® Hierarchy Editor lets you view many levels of a single design using either a table view or a tree view. The Hierarchy Editor also lets you create a configuration that provides expansion information for mixed-signal partitioning. The configuration file is the `expand.cfg` file, which specifies how a design is to be expanded.

You can use the Hierarchy Editor to

- Display the configuration in order to traverse the hierarchy of your design
- Display default cell and instance bindings
- Change global library, view, and stop lists
- Change cell and instance bindings
- Change the inherited view list and library list for cells and instances
- Specify occurrence bindings
- Create new configurations

Starting the Hierarchy Editor

You can start the Hierarchy Editor from a terminal window, from the Virtuoso design environment, and from Cadence applications such as the Virtuoso schematic editor.

Starting the Hierarchy Editor from a Terminal Window

To start the Hierarchy Editor from a terminal window,

➤ Type the following command:

```
cdsHierEditor &
```

For a list of options with which you can start the Hierarchy Editor, see “Command-line Options” on page 21.

Starting the Hierarchy Editor from the Virtuoso Design Environment

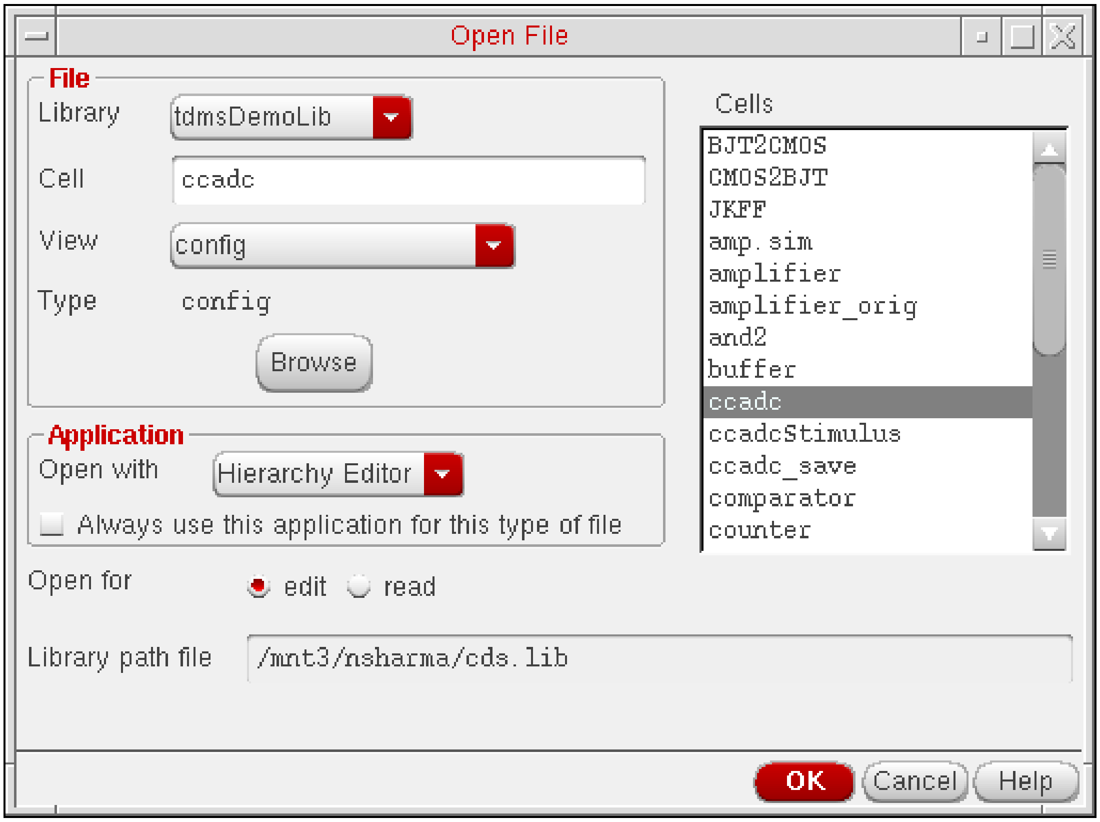
To start the Hierarchy Editor from the Virtuoso design environment,

1. Start the Virtuoso design environment.

```
virtuoso &
```

2. Choose *File – Open*.

   The Open File form appears.



Open File

File

Library: tdmsDemoLib

Cell: ccadc

View: config

Type: config

Application

Open with: Hierarchy Editor

□ Always use this application for this type of file

Open for: ● edit ○ read

Library path file: /mnt3/nsharma/cds.lib

Cells

BJT2CMOS
CMOS2BJT
JKFF
amp.sim
amplifier
amplifier_orig
and2
buffer
ccadc
ccadcStimulus
ccadc_save
comparator
counter

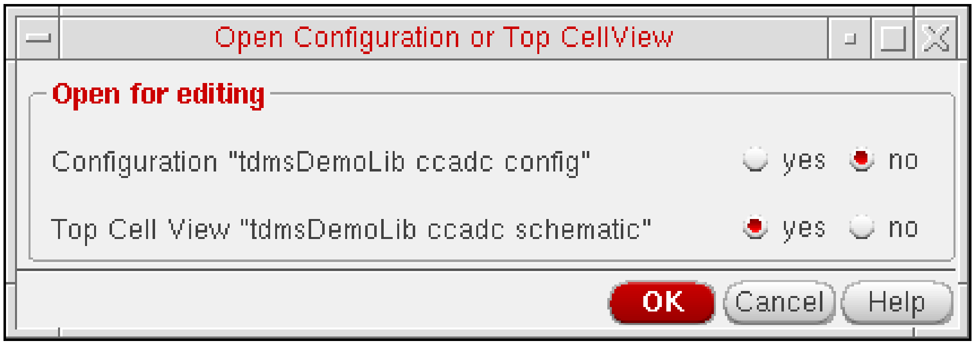
3. In the *File* section of the form, select the library, cell, and view of the configuration that you want to open.

4. In the *Application* field, set the application to *Hierarchy Editor*, if it is not already set.

5. In the *Open for* field, select *edit* if you want to edit the configuration or *read* if you want to open the configuration read-only.

6. Click *OK*.

The Open Configuration or Top Cellview form appears.



Configuration "tdmsDemoLib ccadc config" ○ yes ● no

Top Cell View "tdmsDemoLib ccadc schematic" ● yes ○ no

7. In the form, in the *Configuration "configName"* field, select *yes*. This selection will open the Hierarchy Editor.

8. In the *Top Cell View "top cellview Name"* field, select *yes* if you want to open the schematic editor also or *no* if you do not want to open the schematic editor.

9. Click *OK*.

The Hierarchy Editor opens and displays the configuration. If you chose yes in the *Top Cell View* field, the schematic editor also opens and displays the top cellview of the configuration.

Starting the Hierarchy Editor from Other Applications

You can also start the Hierarchy Editor from other Cadence applications such as the schematic editor. For more information, see the application's documentation.

Command-line Options

When you start the Hierarchy Editor from the command line, you can specify the following options.

Syntax

```
cdsHierEditor
     [-h[elp]]
     [-cdslib filePath]
     [-lib libname]
     [-cell cellname]
     [-view viewname]
     [-mode {r|w|a}]
     [-namespace nmpName]
     [-cdslib filepath]
     [-log filename]
     [-restore dirname]
     [-tree]
     [-version]
     [-ignoreRootConfig]
     [-plugin pluginName [pluginOptions ...]]
```

Note: If you use the `-plugin` option, you must specify it as the last argument. The Hierarchy Editor ignores all arguments after `-plugin`, except another `-plugin` argument.

Arguments

`-h[elp]`	Prints a list and a brief description of the command-line options.
`-cdslib` *`filePath`*	Specifies the `cds.lib` file to load. The `cds.lib` file defines the libraries that you can use. If you do not specify this option, the Cadence Search File mechanism (CSF) is used to find the `cds.lib` file to load.
`-lib` *`libname`*	Specifies the library name of the configuration to be opened.
`-cell` *`cellname`*	Specifies the cell name of the configuration to be opened.

`-view` *`viewname`*

Specifies the view name of the configuration to be opened. You must specify all three parts of the `-lib`, `-cell`, and `-view` options for the command to be complete.

`-mode {r|w|a}`

Specifies the editing mode (read, write, or append) in which the Hierarchy Editor opens the configuration. The default mode is append.

Note: The `-mode w` (write) option erases all data in the configuration file you specify and opens the empty configuration file.

`-namespace` *`nmpName`*

Specifies the name space to operate in, for example, `Verilog`, or `VHDL`. All names are then displayed in that name space, regardless of the source description that is used.

`-cdslib` *`filepath`*

Specifies the `cds.lib` file to load. The `cds.lib` file specifies the libraries to use.

`-log` *`filename`*

By default the Hierarchy Editor creates a log file called `hierEditor.log` to record commands and messages from the session. If that file is locked, it creates `hierEditor.log.1`, and if that is also locked, `hierEditor.log.2`, and so on, until `hierEditor.log.10`.

Use the `-log` *`filename`* option if you want to specify another log file name and path.

`-restoreDir` *`dirname`*

Loads the `hed.env` file from the specified directory after other Hierarchy Editor files (found with the Cadence Search File mechanism) have been loaded. The `hed.env` file contains saved environment settings.

`-tree`

Specifies that the tree view is to be displayed at startup. If this argument is not specified, the table view is displayed.

`-version`

Prints version information and exits. The `-version` option displays the following version information:

- Version number of the Hierarchy Editor
- The versions of input files the Hierarchy Editor supports. Examples of input files include `expand.cfg` and `pc.db`.
- The version of output files the Hierarchy Editor outputs. Examples of output files include `expand.cfg` and Verilog.

For example:

```
Tool:   cdsHierEditor   05.01.000-b005
Input:  expand.cfg      04.04.003
Input:  expand.cfg      05.00.000
Input:  pc.db           01.00
Output: expand.cfg      05.00.000
Output: Verilog         1364-1995
Output: VHDL            1076-1993
```

`-ignoreRootConfig`

Ignores the root configuration in select and highlight messages between the Hierarchy Editor and Virtuoso® Design Environment applications.

By default, if you are not editing the same root configuration in your Virtuoso application and the Hierarchy Editor, select and highlight messages between the two applications are ignored. If you set the `-ignoreRootConfig` option, the Hierarchy Editor ignores the root configuration.

-plugin *pluginName*
[*pluginOptions* ...]

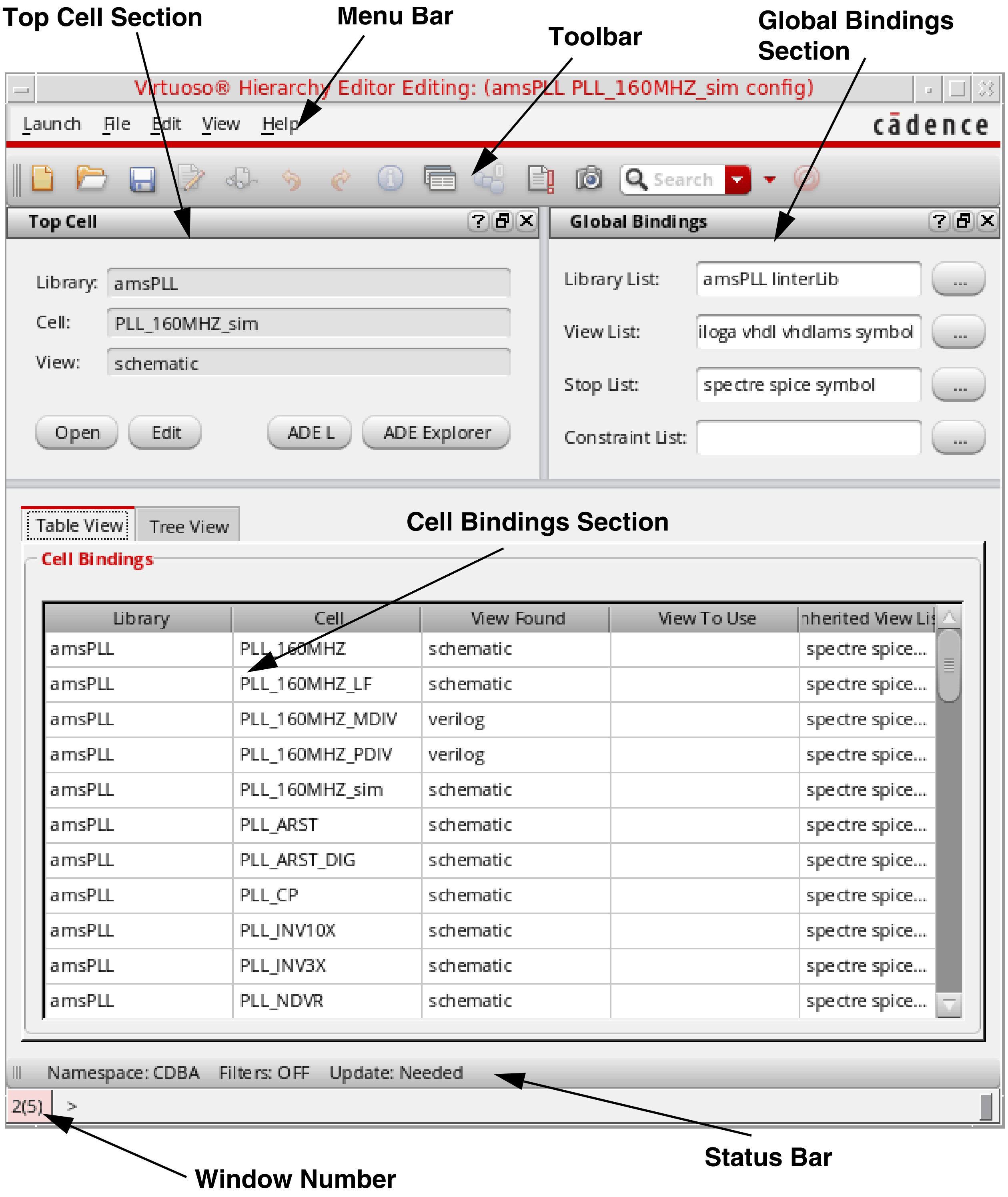
Loads the specified plug-in. You can specify command-line options for the plug-in. To load more than one plug-in, type `-plugin` before each plug-in name.

If you use the `-plugin` option, you must specify it as the last argument. The Hierarchy Editor ignores all arguments after `-plugin`, except another `-plugin` argument.

For more information about plug-ins, see Chapter 5, “Using Plug-Ins.”

Note: Plug-ins must be installed in the *your_install_dir*/share/cdssetup/hierEditor/plugins directory.

About the Hierarchy Editor User Interface



Virtuoso® Hierarchy Editor Editing: (amsPLL PLL_160MHZ_sim config)

Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

Library List: amsPLL linterLib
View List: iloga vhdl vhdlams symbol
Stop List: spectre spice symbol
Constraint List:

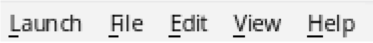
Library	Cell	View Found	View To Use	nherited View Lis
amsPLL	PLL_160MHZ	schematic		spectre spice...
amsPLL	PLL_160MHZ_LF	schematic		spectre spice...
amsPLL	PLL_160MHZ_MDIV	verilog		spectre spice...
amsPLL	PLL_160MHZ_PDIV	verilog		spectre spice...
amsPLL	PLL_160MHZ_sim	schematic		spectre spice...
amsPLL	PLL_ARST	schematic		spectre spice...
amsPLL	PLL_ARST_DIG	schematic		spectre spice...
amsPLL	PLL_CP	schematic		spectre spice...
amsPLL	PLL_INV10X	schematic		spectre spice...
amsPLL	PLL_INV3X	schematic		spectre spice...
amsPLL	PLL_NDVR	schematic		spectre spice...

Namespace: CDBA Filters: OFF Update: Needed

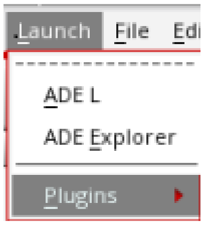
2(5) >

Menus

The menu bar displays the Hierarchy Editor menus.



Launch Menu



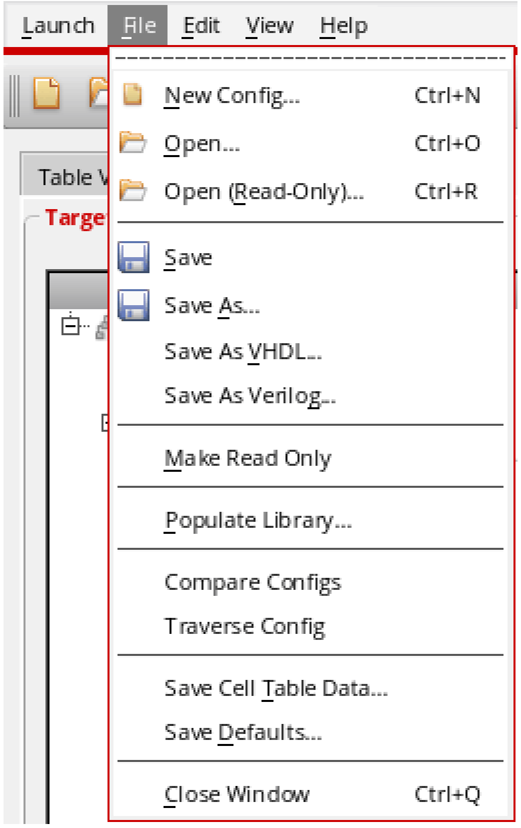
Launch – ADE L opens the ADE L window. (IC6.1.8 Only)

Launch – ADE Explorer opens an existing ADE Explorer view or creates a new ADE Explorer view.

Launch – Plugins includes plugins that are currently registered with Hierarchy Editor. Selecting an option from this menu loads the plugin.

Note: This submenu is not displayed if the plugins are not available.

File Menu



File – New Config creates a new configuration.

File – Open opens an existing configuration for editing.

File – Open (*Read-Only*) opens a configuration for viewing.

File – Save saves the open configuration.

File – Save As saves the open configuration under a new name.

File – Save As VHDL saves the open configuration as a VHDL file.

File – Save As Verilog saves the open configuration as a Verilog® file.

File – Populate Library brings Verilog modules into a library.

File – Compare Configs compares two configurations and highlight their differences by opening the resulting design hierarchy side-by-side in a tree view.

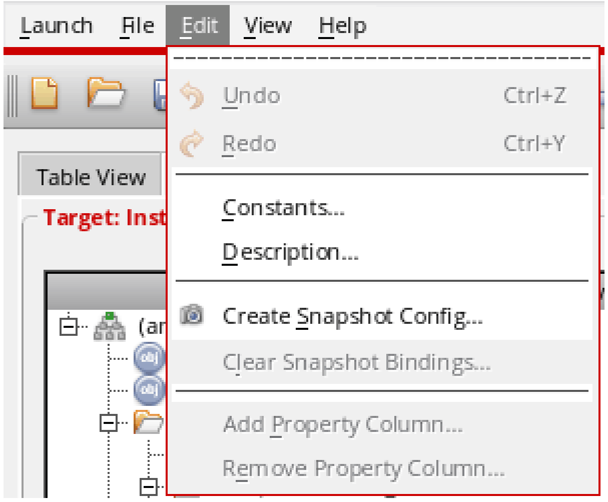
File – Traverse Config traverses through a configuration and save the traversal results in an output file.

File – Save Cell Table Data saves data from the cell table into a text file.

File – Save Defaults saves your settings.

File – Close Window closes the Hierarchy Editor.

Edit Menu



Edit – Undo removes the last action. You can undo an unlimited number of actions.

Edit – Redo redoes the previously undone action.

Edit – Constants opens the Edit Constants form to let you create or edit constants.

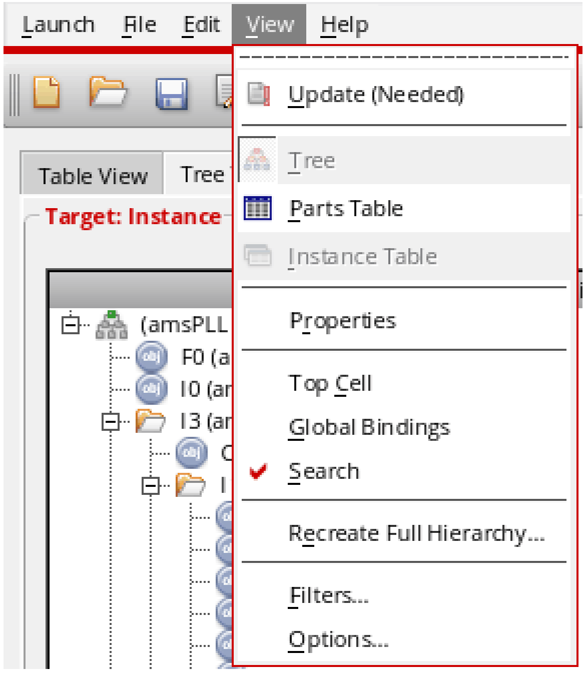
Edit – Description opens the Edit Description form to let you edit the description of the configuration.

Edit – Add Property Column adds a column for a simulation-control property. This menu option is available only if the *View – Properties* option is selected.

Edit – Remove Property Column removes a property column that is currently displayed.

View Menu

View menu commands let you set the display options for the Hierarchy Editor. You can choose to display or hide the tree view or table view, instance table, properties, top cell section, global bindings section, and message area in the Hierarchy Editor window.



View – Update recomputes the hierarchy based on the rules you entered. This command is unavailable until you have opened a configuration.

> **Note:** You can also select the *Automatic Update* option, which is available in the Options form, to automatically update your configuration every time you make a change.

View – Tree displays cell and instance bindings in a tree structure.

View – Parts Table displays cell and instance bindings in a table structure.

View – Instance Table displays the instance table. This option is only available when *View – Parts Table* is selected.

View – Properties displays property columns that let you set simulation-control properties. This menu option is available only if there is a property dictionary in your Cadence installation hierarchy.

View – Top Cell displays or hides the top cell information.

View – Global Bindings displays or hides the global library, view, and stop lists.

View – Search opens the interactive Search assistant pane that offers additional search viewing information compared to that offered by the HED toolbar.

View – Recreate Full Hierarchy recomputes the full hierarchy and recreates all `pc.db` files for any text views in the design.

View – Filters opens the Filters form to let you set or edit display filters.

View – Options opens the Options form to let you select the columns you want to display in the Hierarchy Editor. The Options form also lets you set the *Automatic Update* option.

You can save the display options to use when you restart the Hierarchy Editor. See “Saving Custom Settings” on page 78 for more information.

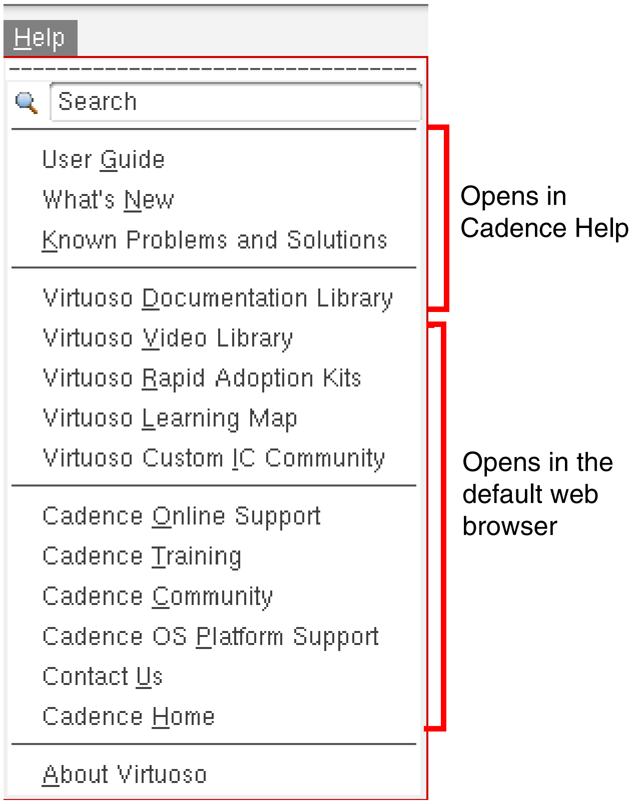
Important

If you access the standalone version of the Hierarchy Editor, there will also *View – Message Area* option. This is not however required in the Virtuoso-integrated version, where the Hierarchy Editor utilizes the CIW message area.

Help and Support Facilities

For help with various Virtuoso products, do the following:

1. Choose *Help*.



Opens in Cadence Help

Opens in the default web browser

2. Choose one of the following menu options:

Help menu option	Description
Search	A text field that lets you enter a search string. Press `Enter` to view the search results. **Note:** Do not enclose the search string in double quotes.
User Guide	Opens the guide in Cadence Help.
What’s New	Opens the Virtuoso What’s New document in Cadence Help.
Known Problems and Solutions	Opens the Virtuoso Known Problems and Solutions document in Cadence Help.

Virtuoso Documentation Library

Opens the Cadence Help home page, which provides quick access links to the following local and online resources:

- What’s New
- Video Demos and Tutorials
- Featured Content
- Known Problems and Solutions
- Other web resources

Virtuoso Video Library

Opens the Video Library page available on Cadence Online Support (COS). This page lists the videos available for various Virtuoso products.

Note: You must have a COS account to access the content available on COS.

Note: Contact your IT support to ensure that the Internet ports required for video playback are enabled.

Virtuoso Rapid Adoption Kits

Opens the Rapid Adoption Kits page on COS. This page lists Rapid Adoption Kits (RAKs) available for various Virtuoso products. A RAK contains design databases and instructions on how to run the design flow.

Virtuoso Learning Map

Lists domain-specific training available on Cadence Training Services.

Cadence Training Services learning maps provide a comprehensive visual overview of the learning opportunities for Cadence customers. They provide recommended course flows as well as tool experience and knowledge levels to guide customers through a complete learning plan.

Virtuoso Custom IC Community

Opens the Virtuoso Custom IC Community web page. This page provides access to the latest blogs and discussion threads on various Virtuoso products and design topics, information about software downloads and support and training, and other related information. You too can contribute to the community forum by creating a Cadence account. This gives you additional benefits such as alerts about topics of interest and access to online webinars.

Cadence Online Support

Opens COS, which you can use to access information about Cadence products, documentation, videos, RAKs, application notes, troubleshooting information, alerts, and so on. Improvements are regularly made to COS to make it simple for you to look up the information you want. We recommend that you bookmark this web site and use it as your first point of reference for any Virtuoso-related information.

Tip

You can also access COS by clicking the Cadence logo available in the upper-right banner in each Virtuoso window.

Cadence Training

Opens the Cadence training web page. You can find on this page information about the training courses available in different regions. Information is available about both classroom and online courses.

Cadence Community

Opens the Cadence Community web page. This page provides access to the latest blogs and discussion threads on various Cadence products and solutions, and EDA Industry Insights. You too can contribute to the community forum by creating a Cadence account. This gives you additional benefits such as alerts about topics of interest and access to online webinars.

Cadence OS Platform Support

Provides information about the current Cadence software releases and the supported platforms.

Contact Us

Opens the Cadence Customer Support web page, which provides customer support contact information for different regions.

Cadence Home

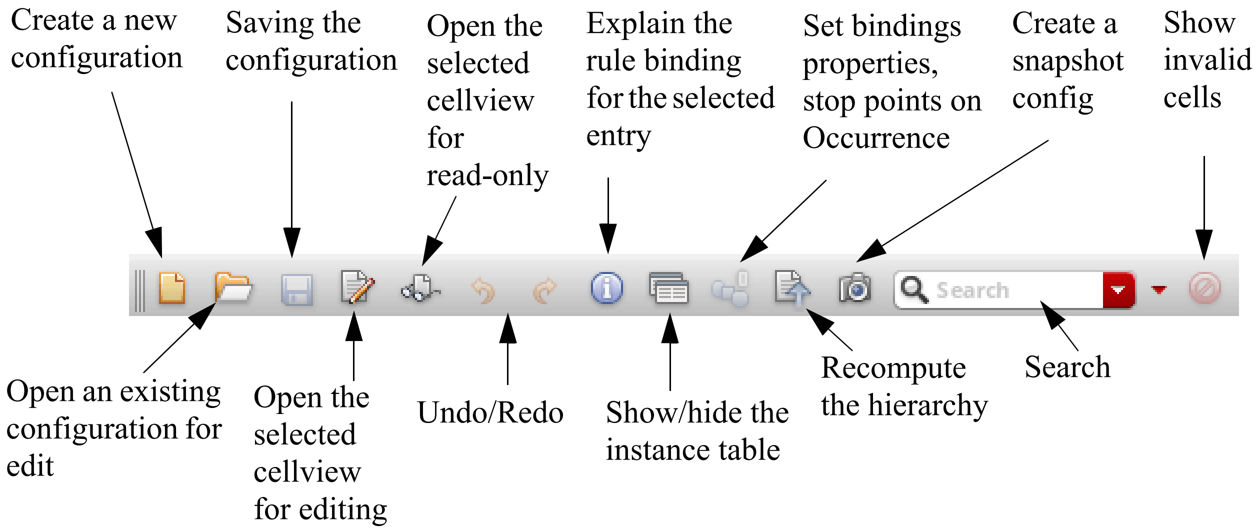
Opens the Cadence corporate web site.

About Virtuoso

Displays Virtuoso Design Environment version information.

HED Toolbar

You can access frequently used commands from the toolbar.


Create a new configuration
Saving the configuration
Open the selected cellview for read-only
Explain the rule binding for the selected entry
Set bindings properties, stop points on Occurrence
Create a snapshot config
Show invalid cells
Search
Open an existing configuration for edit
Open the selected cellview for editing
Undo/Redo
Show/hide the instance table
Recompute the hierarchy
Search

You can move the toolbar to any side of the main window. You cannot float it.

To move the toolbar,

➤ Drag it by the handle and place it on any inside edge of the main window.

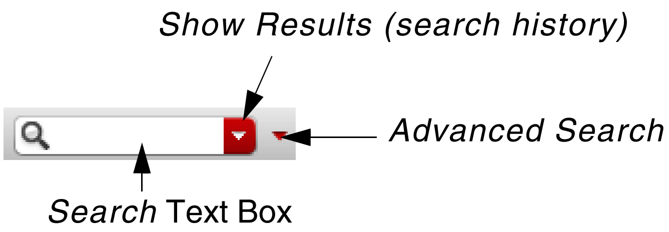
Note: There is an *Invalid Cell* icon, which will become active if there are any invalid (unbound) cells when a re-elaboration is done. If there are any unbound cells, clicking the icon will update the cell table to show those invalid cells at the top of the table.

Using the Search Functionality

The *Search* functionality in HED comprises of two search entry mechanisms:

- The HED Toolbar
- The Search Assistant Pane

The HED Toolbar



The search functionality on the HED toolbar consists of the following UI components:

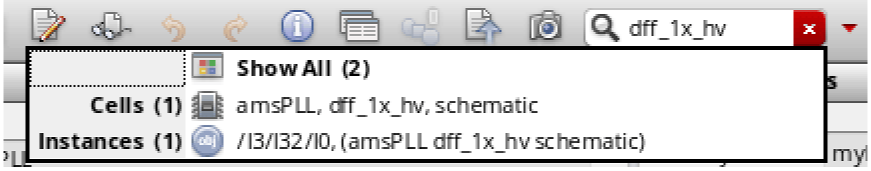
- The Search Text Box
- Show Results Drop-down List
- Advanced Search

The Search Text Box

The *Search* text box is used to enter any search queries, such as the cell name, instance name, or occurrence path in the Virtuoso® Hierarchy Editor window.

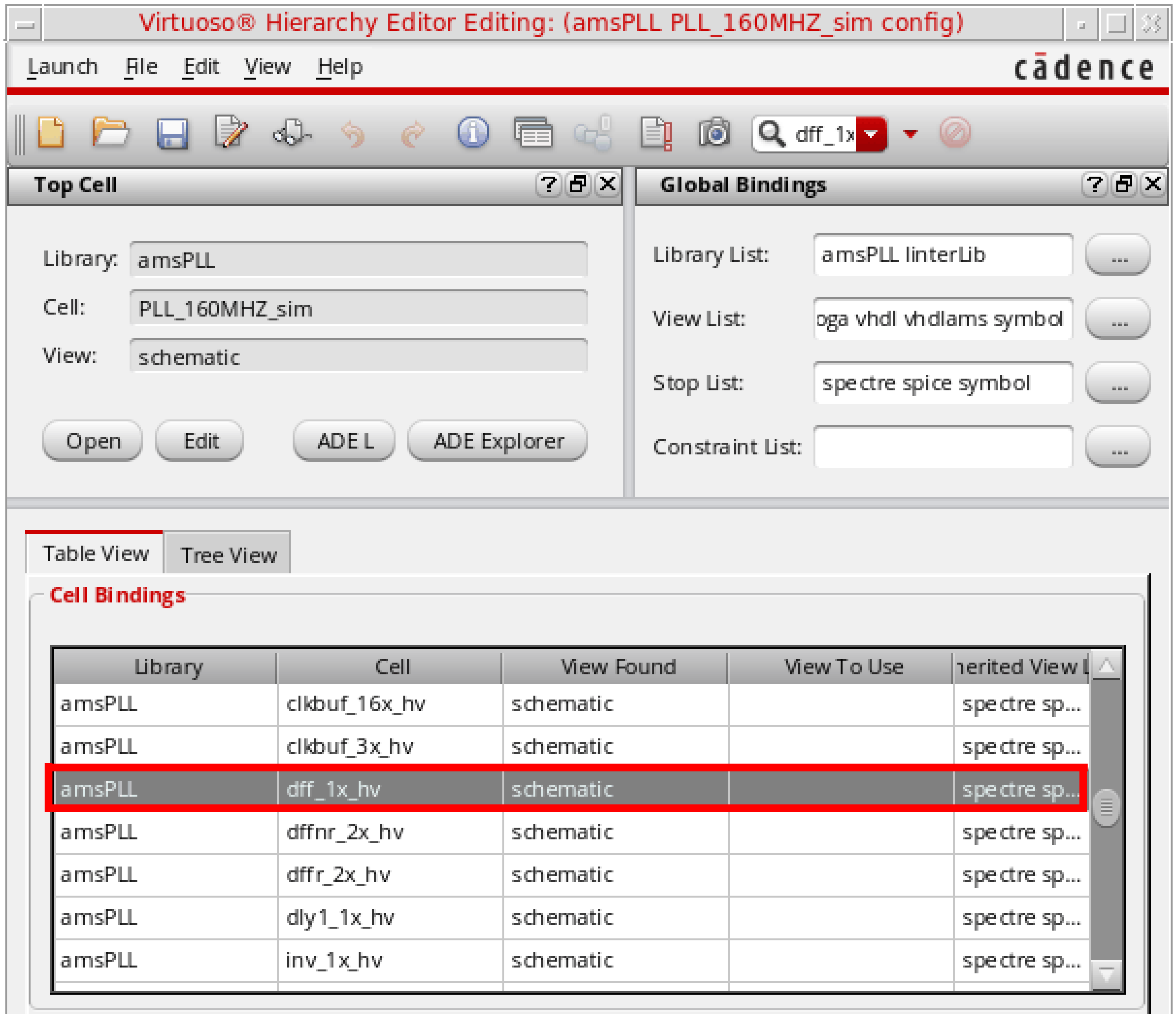
Note: If you type a string in the *Search* text box that begins with the same set of characters as those contained at the beginning of a search string entered earlier, the string entered earlier is automatically displayed as a choice (auto-complete).

Once you type the search queries in the *Search* text box and press *Enter*, the results are displayed immediately in the *Show Results* drop-down list.



Cells (1) amsPLL, dff_1x_hv, schematic
Instances (1) /I3/I32/I0, (amsPLL dff_1x_hv schematic)

On selecting the cell from the *Show Results* drop-down list, the corresponding entry is highlighted in the *Table View*.

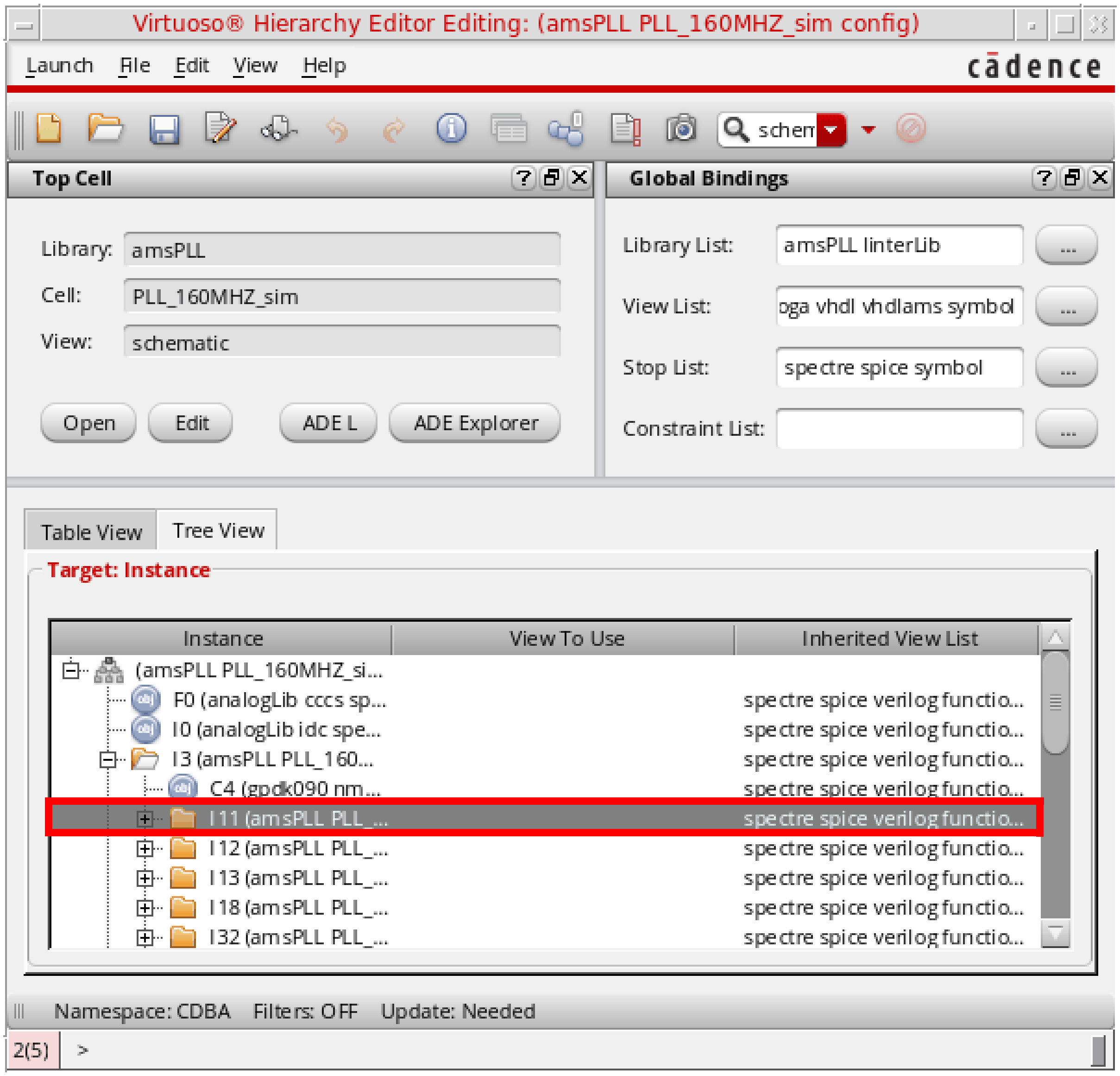


Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

Library List: amsPLL linterLib
View List: oga vhdl vhdlams symbol
Stop List: spectre spice symbol
Constraint List:

Library	Cell	View Found	View To Use	ıerited View l
amsPLL	clkbuf_16x_hv	schematic		spectre sp...
amsPLL	clkbuf_3x_hv	schematic		spectre sp...
amsPLL	dff_1x_hv	schematic		spectre sp...
amsPLL	dffnr_2x_hv	schematic		spectre sp...
amsPLL	dffr_2x_hv	schematic		spectre sp...
amsPLL	dly1_1x_hv	schematic		spectre sp...
amsPLL	inv_1x_hv	schematic		spectre sp...

Similarly, on selecting the instance or occurrence path from the *Show Results* drop-down list, the corresponding entry is highlighted in the *Tree View*.



Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

Library List: amsPLL linterLib
View List: oga vhdl vhdlams symbol
Stop List: spectre spice symbol
Constraint List:

Target: Instance

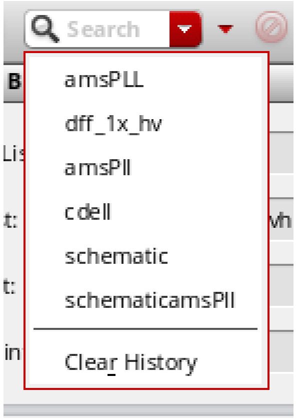
Instance	View To Use	Inherited View List
(amsPLL PLL_160MHZ_si...		
F0 (analogLib cccs sp...		spectre spice verilog functio...
I0 (analogLib idc spe...		spectre spice verilog functio...
I3 (amsPLL PLL_160...		spectre spice verilog functio...
C4 (gpdk090 nm...		spectre spice verilog functio...
I11 (amsPLL PLL_...		spectre spice verilog functio...
I12 (amsPLL PLL_...		spectre spice verilog functio...
I13 (amsPLL PLL_...		spectre spice verilog functio...
I18 (amsPLL PLL_...		spectre spice verilog functio...
I32 (amsPLL PLL_...		spectre spice verilog functio...

Namespace: CDBA Filters: OFF Update: Needed

2(5) >

Show Results Drop-down List

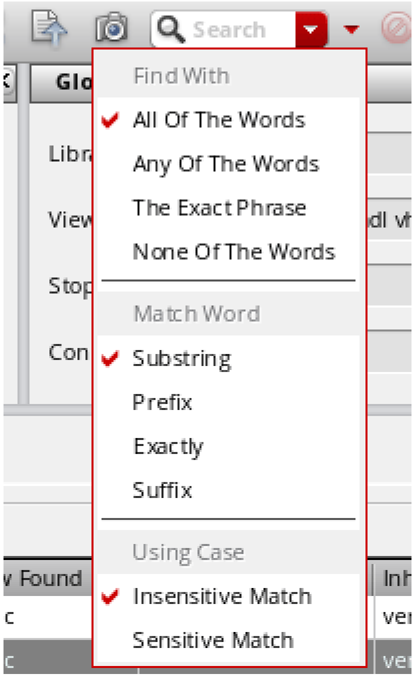
Clicking the *Show Results* drop-down list will display a recent search history list where you can immediately recall the results of any recent searches and reinstate any of the search options selected when that search criteria was last applied.



Note: To clear the search history, you need to select the *Clear History* option from the *Show Results* drop-down list.

Advanced Search

The *Advanced Search* drop-down list is located to the right of *Search* drop-down list, and provides the following commonly used search options to refine your search results.



- The options under the *Find With* section are used to specify the default matching operators `AND` (*All Of The Words*), `OR` (*Any Of The Words*), `EXACTLY` (*The Exact Phrase*), and `NOT` (*None Of The Words*).

  Default option: *All Of The Words*

  Also see Query in the *Virtuoso Schematic Editor* User Guide.

- The options under the *Match Word* section let you specify a regular query token that must be matched in the current content search data to display a successful result. The options available here are *Prefix*, *Substring*, *Exactly*, and *Suffix*.

  Default option: *Substring*

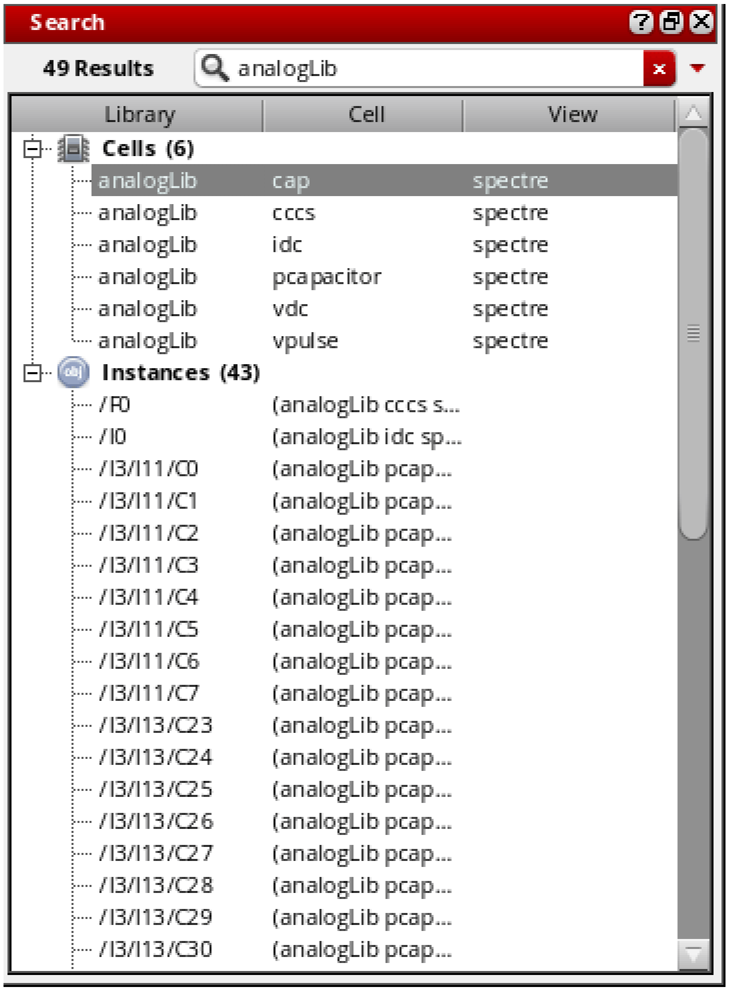
  Also see Word in the *Virtuoso Schematic Editor* User Guide.

- The options under *Using Case* let you determine whether search results must be case sensitive (*Sensitive Match*) or whether any text case is acceptable (*Insensitive Match*).

  Default option: *Insensitive Match*

The Search Assistant Pane

The interactive *Search* assistant pane is a dockable/undockable pane that offers additional search viewing information compared to that offered by the HED toolbar.



49 Results

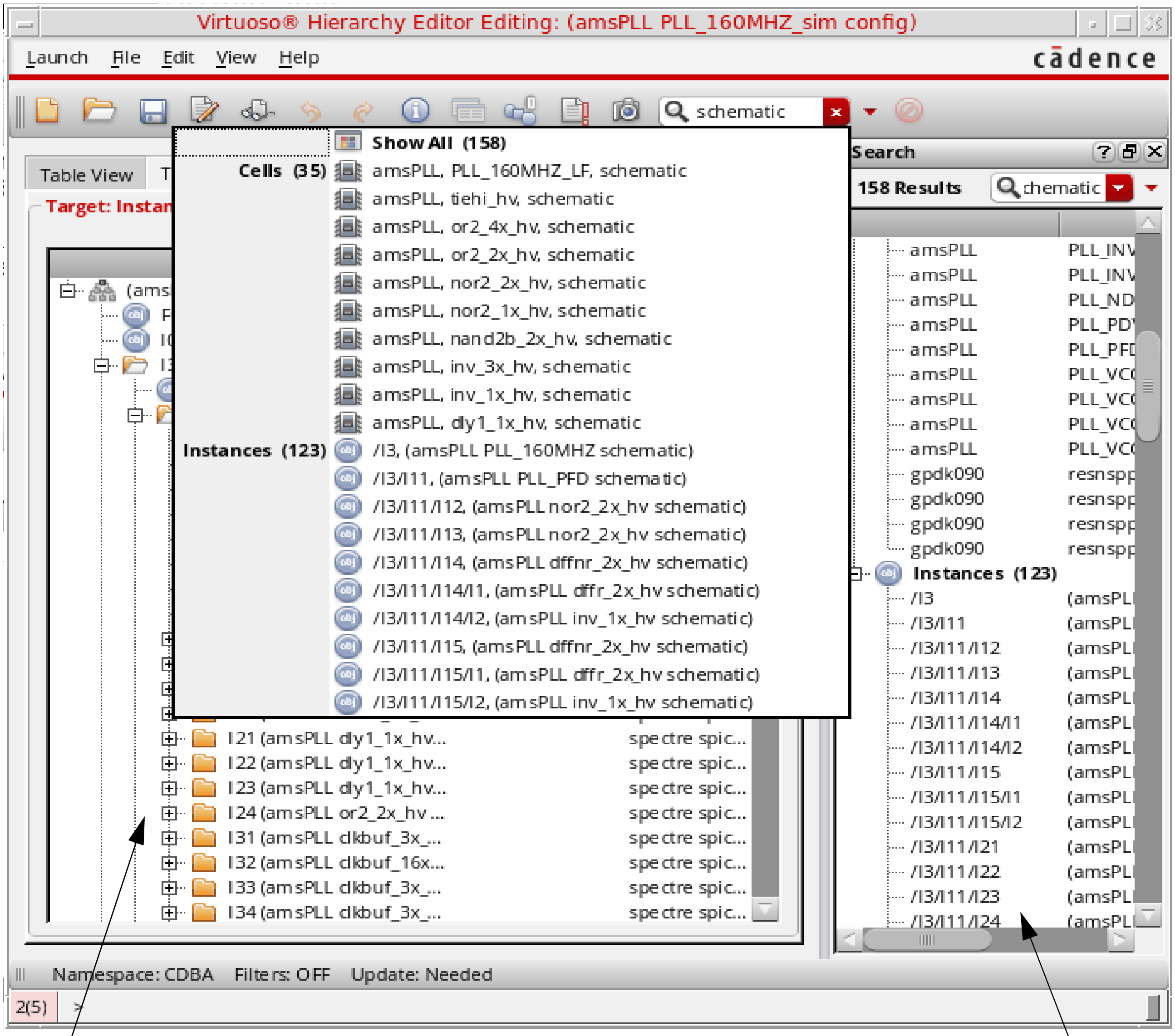
analogLib

Library	Cell	View
Cells (6)		
analogLib	cap	spectre
analogLib	cccs	spectre
analogLib	idc	spectre
analogLib	pcapacitor	spectre
analogLib	vdc	spectre
analogLib	vpulse	spectre
Instances (43)		
/F0	(analogLib cccs s...	
/I0	(analogLib idc sp...	
/I3/I11/C0	(analogLib pcap...	
/I3/I11/C1	(analogLib pcap...	
/I3/I11/C2	(analogLib pcap...	
/I3/I11/C3	(analogLib pcap...	
/I3/I11/C4	(analogLib pcap...	
/I3/I11/C5	(analogLib pcap...	
/I3/I11/C6	(analogLib pcap...	
/I3/I11/C7	(analogLib pcap...	
/I3/I13/C23	(analogLib pcap...	
/I3/I13/C24	(analogLib pcap...	
/I3/I13/C25	(analogLib pcap...	
/I3/I13/C26	(analogLib pcap...	
/I3/I13/C27	(analogLib pcap...	
/I3/I13/C28	(analogLib pcap...	
/I3/I13/C29	(analogLib pcap...	
/I3/I13/C30	(analogLib pcap...	

The *Search* assistant pane is displayed in either of the following cases:

- Selecting the *View – Search* option from the Virtuoso® Hierarchy Editor window.
- Clicking the *Show All* option from the *Show Results* drop-down menu.
- Clicking the Right Mouse Button on the entry in the Table View and selecting the *Search Hierarchy For* option from the context-sensitive menu.
- Clicking the Right Mouse Button on the entry in the *Instantiations* list box of the Explain form and selecting the *Search Hierarchy For* option from the context-sensitive menu.

Note: As compared to the HED toolbar search, which shows only the top `10` matching results, the *Search* assistant pane shows all the matching results.



Show All (158)

Cells (35)
- amsPLL, PLL_160MHZ_LF, schematic
- amsPLL, tiehi_hv, schematic
- amsPLL, or2_4x_hv, schematic
- amsPLL, or2_2x_hv, schematic
- amsPLL, nor2_2x_hv, schematic
- amsPLL, nor2_1x_hv, schematic
- amsPLL, nand2b_2x_hv, schematic
- amsPLL, inv_3x_hv, schematic
- amsPLL, inv_1x_hv, schematic
- amsPLL, dly1_1x_hv, schematic

Instances (123)
- /I3, (amsPLL PLL_160MHZ schematic)
- /I3/I11, (amsPLL PLL_PFD schematic)
- /I3/I11/I12, (amsPLL nor2_2x_hv schematic)
- /I3/I11/I13, (amsPLL nor2_2x_hv schematic)
- /I3/I11/I14, (amsPLL dffnr_2x_hv schematic)
- /I3/I11/I14/I1, (amsPLL dffr_2x_hv schematic)
- /I3/I11/I14/I2, (amsPLL inv_1x_hv schematic)
- /I3/I11/I15, (amsPLL dffnr_2x_hv schematic)
- /I3/I11/I15/I1, (amsPLL dffr_2x_hv schematic)
- /I3/I11/I15/I2, (amsPLL inv_1x_hv schematic)

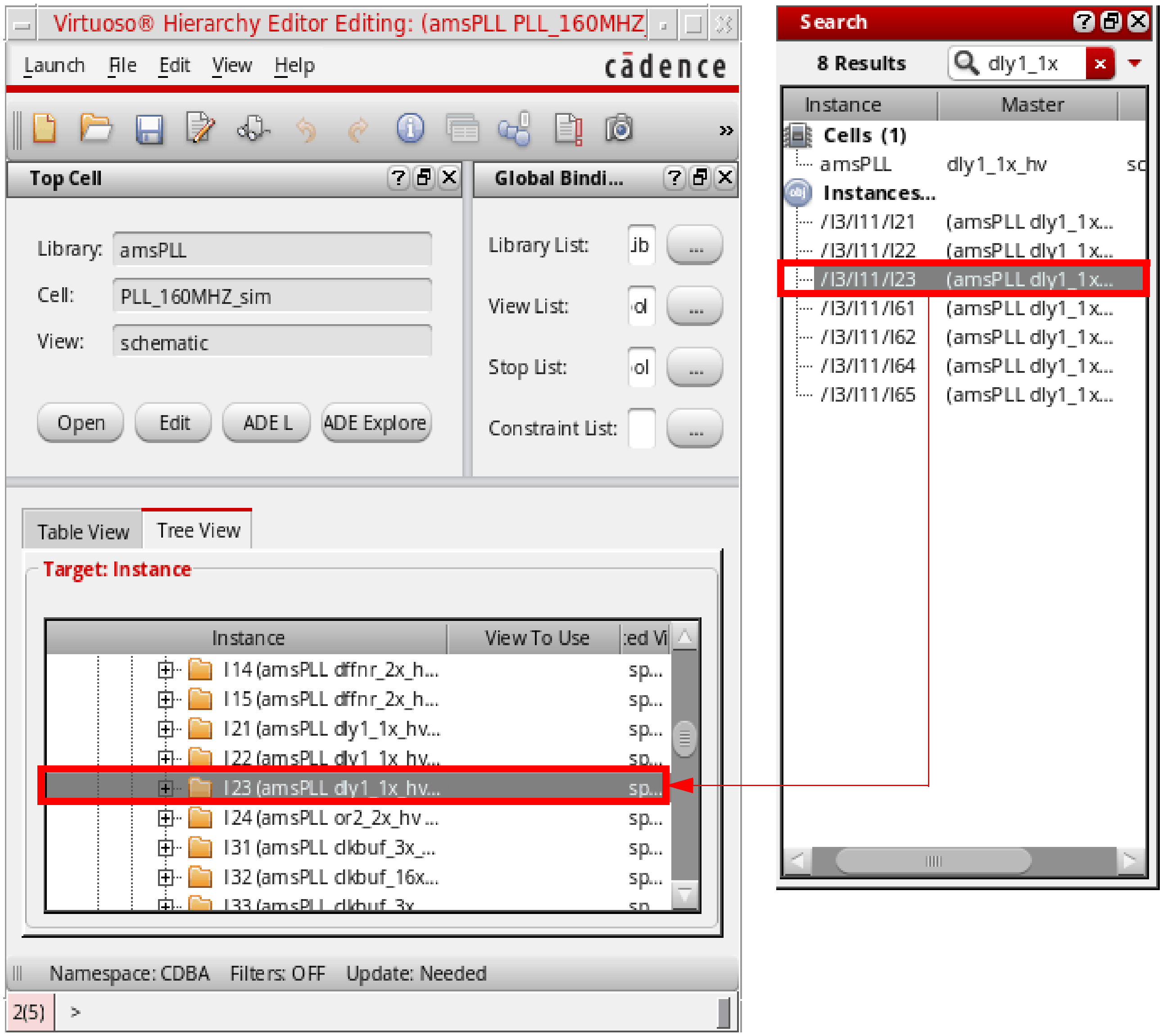
I21 (amsPLL dly1_1x_hv...	spectre spic...
I22 (amsPLL dly1_1x_hv...	spectre spic...
I23 (amsPLL dly1_1x_hv...	spectre spic...
I24 (amsPLL or2_2x_hv ...	spectre spic...
I31 (amsPLL clkbuf_3x_...	spectre spic...
I32 (amsPLL clkbuf_16x...	spectre spic...
I33 (amsPLL clkbuf_3x_...	spectre spic...
I34 (amsPLL clkbuf_3x_...	spectre spic...

amsPLL	PLL_INV
amsPLL	PLL_INV
amsPLL	PLL_ND
amsPLL	PLL_PD
amsPLL	PLL_PFD
amsPLL	PLL_VC
amsPLL	PLL_VC
amsPLL	PLL_VC
amsPLL	PLL_VC
gpdk090	resnsp
gpdk090	resnsp
gpdk090	resnsp
gpdk090	resnsp
Instances (123)	
/I3	(amsPLL
/I3/I11	(amsPLL
/I3/I11/I12	(amsPLL
/I3/I11/I13	(amsPLL
/I3/I11/I14	(amsPLL
/I3/I11/I14/I1	(amsPLL
/I3/I11/I14/I2	(amsPLL
/I3/I11/I15	(amsPLL
/I3/I11/I15/I1	(amsPLL
/I3/I11/I15/I2	(amsPLL
/I3/I11/I21	(amsPLL
/I3/I11/I22	(amsPLL
/I3/I11/I23	(amsPLL
/I3/I11/I24	(amsPLL

Shows top 10 matching Cells and Instances

Shows all matching Cells and Instances

On selecting the cell or instance in the *Search* assistant pane, the corresponding entry is highlighted in the *Tree View* or *Table View* of the HED window.



Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

Library List: ib
View List: ol
Stop List: ol
Constraint List:

Target: Instance

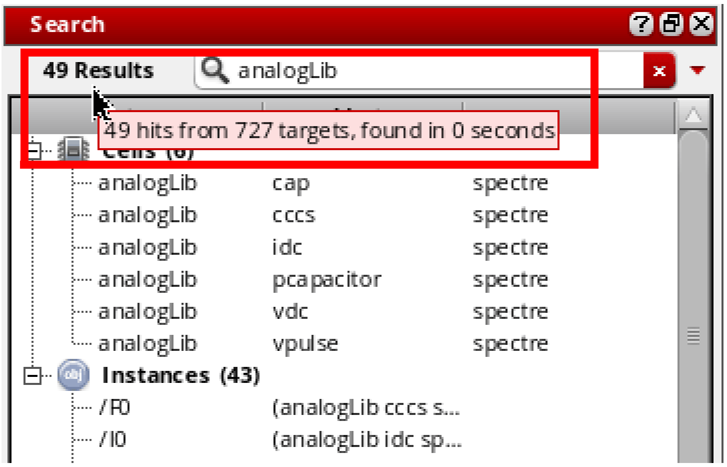
Instance	View To Use	:ed Vi
I14 (amsPLL dffnr_2x_h...		sp...
I15 (amsPLL dffnr_2x_h...		sp...
I21 (amsPLL dly1_1x_hv...		sp...
I22 (amsPLL dly1_1x_hv...		sp...
I23 (amsPLL dly1_1x_hv...		sp...
I24 (amsPLL or2_2x_hv ...		sp...
I31 (amsPLL clkbuf_3x_...		sp...
I32 (amsPLL clkbuf_16x...		sp...
I33 (amsPLL clkbuf_3x		sp

Namespace: CDBA Filters: OFF Update: Needed

2(5) >

Instance	Master	
Cells (1)		
amsPLL	dly1_1x_hv	sc
Instances...		
/I3/I11/I21	(amsPLL dly1_1x...	
/I3/I11/I22	(amsPLL dly1_1x...	
/I3/I11/I23	(amsPLL dly1_1x...	
/I3/I11/I61	(amsPLL dly1_1x...	
/I3/I11/I62	(amsPLL dly1_1x...	
/I3/I11/I64	(amsPLL dly1_1x...	
/I3/I11/I65	(amsPLL dly1_1x...	

The total number of search hits are displayed at the top-left corner of the *Search* assistant pane. Placing the pointer over this number will also display the potential hit targets and how long it took to perform the current search. For example, "*49 hits from 727 targets, found in 0 seconds*".



analogLib	cap	spectre
analogLib	cccs	spectre
analogLib	idc	spectre
analogLib	pcapacitor	spectre
analogLib	vdc	spectre
analogLib	vpulse	spectre
/F0	(analogLib cccs s...	
/I0	(analogLib idc sp...	

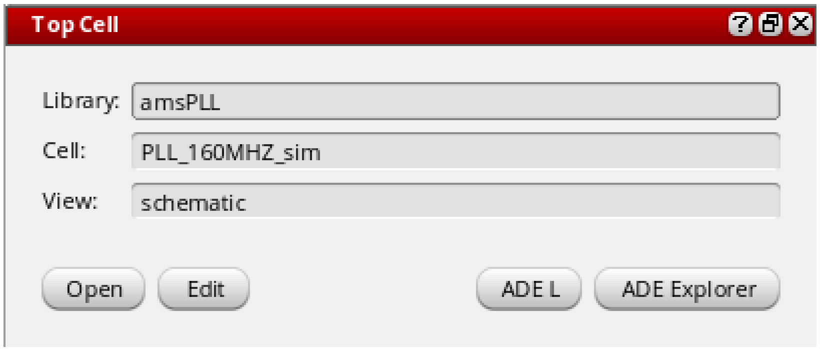
Video

To view the demonstration on the search functionality, see Using the Search Functionality in Hierarchy Editor video.

Note: Access to this video will depend on the availability of a web browser and a Cadence Online Support account.

Top Cell Section

The *Top Cell* section contains the library, cell, and view names of the top cellview of the configuration you opened. This is the root cellview of your design.



Top Cell

Library: amsPLL

Cell: PLL_160MHZ_sim

View: schematic

Library is the library name of the top cellview of the configuration.

Cell is the cell name of the top cellview of the configuration.

View is the view name of the top cellview of the configuration.

Open opens the top cellview in an editor. If the top cellview is a schematic, it is opened in the Virtuoso® schematic editor; if it is a Verilog or VHDL view, it is opened in a text editor.

Edit opens the Edit Top Cell form where you can select alternative *Library*, *Cell*, and *View* selections for the Top Cell from drop-down menus.

ADE L opens the ADE L window. (IC6.1.8 Only)

ADE Explorer opens an existing ADE Explorer view or creates a new ADE Explorer view.

- When opening an existing ADE Explorer view, the simulator is the same as previously set in the cellview.
- When creating a new ADE Explorer view, the simulator is set to "`ams`" by default.

Similar to ADE L, you can disable the default "`ams`" simulator setting either by deselecting the *Set simulator to AMS when launching ADE* check box on the *General* tab of the Options form or by setting the following HED cdsenv variable:

```
hed.ade setAmsSimulator boolean nil
```

Note: After doing this, simulator "`ams`" or "`spectre`" is determined by the following ADE cdsenv variable when a new ADE Explorer view is created:

`asimenv.startup simulator string “ams“/”spectre”` //default is “`spectre`”

Note: Invoking ADE L does not have any effect on the “`asimenv.startup simulator`” cdsenv variable value.

For more information refer to Opening ADE Explorer From CIW in the *Virtuoso ADE Explorer User Guide*.

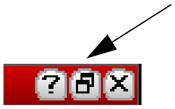
The Top Cell section is a dockable window. You can dock it on any side of the main Hierarchy Editor window or undock it to be a floating window.

To change the location of the Top Cell window,

➤ Drag it by the title bar and place it along any inside edge of the main window.

To float the Top Cell window,

➤ Click the *Float* button on the title bar of the window or double-click the title bar.



You can dock it by clicking the *Float* button again or by double-clicking the title bar.

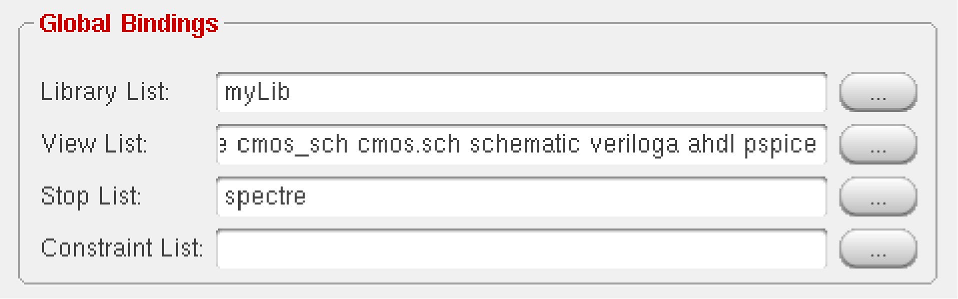
To display or hide the Top Cell window,

➤ Select *View – Top Cell*.

Global Bindings Section

The *Global Bindings* section contains the global library, view, and stop lists. These are the default values for the entire configuration.

Note: Selecting the browse (...) button for a specific *Global Bindings* list option will display a view list building form that will allow you to build up a view list without the need to manually type in the libraries, views, and so on. For more information see View List Building Forms on page 48.



Library List: myLib
View List: e cmos_sch cmos.sch schematic veriloga ahdl pspice
Stop List: spectre
Constraint List:

Library List is a list of libraries, specified in order of preference, that are searched for cells. Each cell is obtained from the first library in the list that contains a cell of that name.

View List is a list of views, specified in order of preference, for cells. For each cell, the first view in the list that is found is used. For example, if the view list is `tbench_1 schematic dataflow`, the netlister first looks for a `tbench_1` view. If it is not found, it looks for a `schematic` view. If that is not found either, it looks for a `dataflow` view.

Stop List is the list of views used to determine when the hierarchy expansion will stop and when a cell should be considered a leaf node in the hierarchy.

Constraint List is a list of constraint views, listed in order of preference. The first view that is found is used. If none of the constraint views in the list are found, the schematic editor creates an empty view named after the first view in the constraint list.

For more information about the rules you can set in the Global Bindings section, see Chapter 3, “Changing Design Components.”

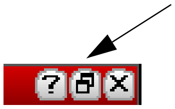
The Global Bindings section is a dockable window. You can dock it along any side of the main Hierarchy Editor window or undock it to be a floating window.

To change the location of the Global Bindings window,

➤ Drag it by the title bar and place it along any inside edge of the main window.

To float the Global Bindings window,

➤ Click the *Float* button on the title bar of the window or double-click the title bar.



You can dock it by clicking the button again or double-clicking the title bar.

To display or hide the Global Bindings window,

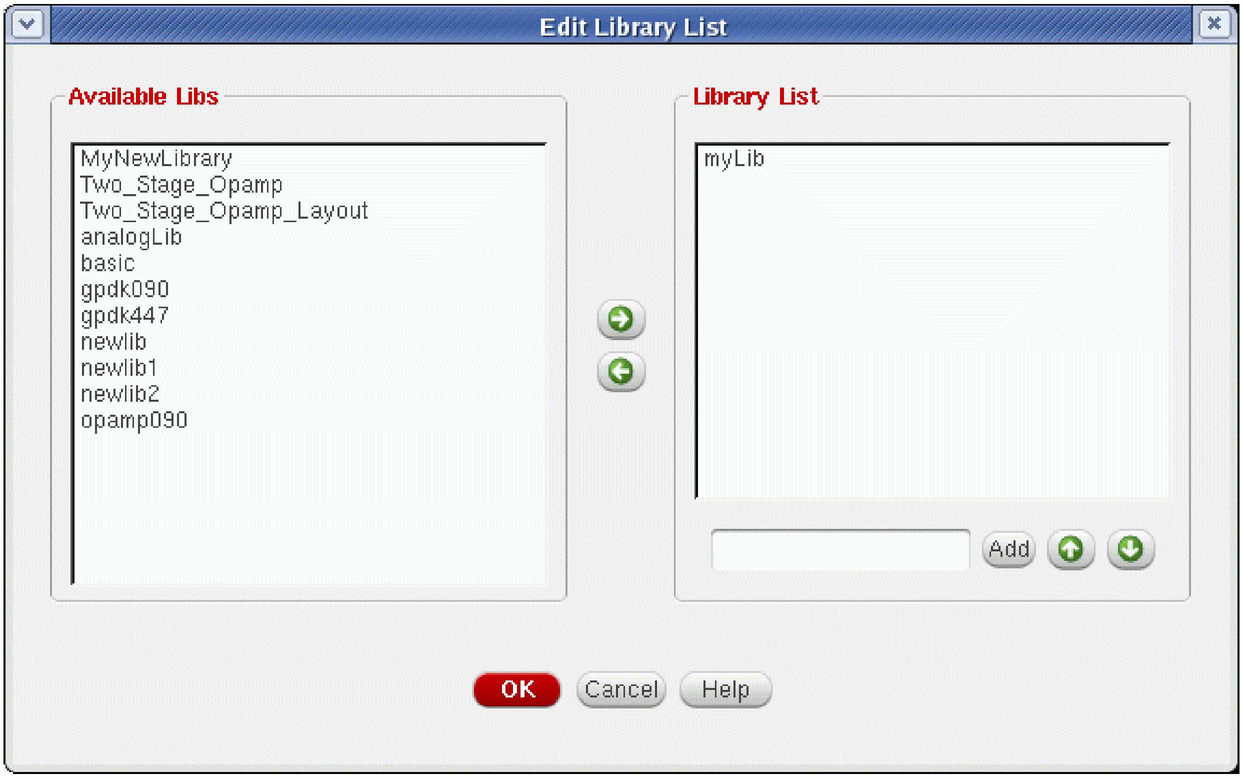
➤ Select *View – Global Bindings*.

View List Building Forms

List builders can be used to modify global bindings in HED, saving on the need to manually create global binding lists.

A view list building form is displayed when you select the browse (...) button for any of the following options in the *Global Bindings* section:

- *Library List* displays the Edit Library List form
- *View List* displays the Edit View List form
- *Stop List* displays the Edit Stop List form
- *Constraint List* displays the Edit Constraint List form



Available Libs:

MyNewLibrary
Two_Stage_Opamp
Two_Stage_Opamp_Layout
analogLib
basic
gpdk090
gpdk447
newlib
newlib1
newlib2
opamp090

Library List:

myLib

A view list building form lets you select from a list of available items (for example *Available Libs*) and move them across, using the *Left* and *Right* green arrow options in the center, to the right-hand side to form part of the global bindings list for libraries, views, and so on.

As well as choosing what you want to include in this list, you can also include further items using the *Add* option and also change the order of the list using the *Up* and *Down* green arrows.

Once you have completed the list updates, click the *OK* button and your updates will be reflected in the appropriate field in the *Global Bindings* section of the HED.

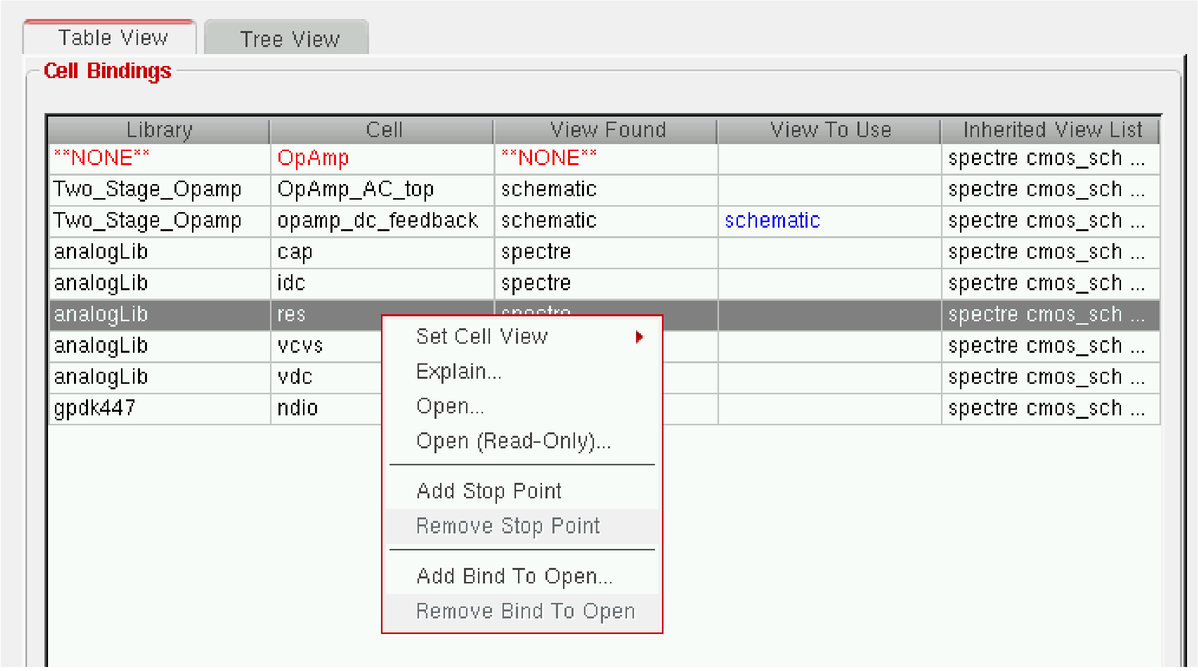
Table View

The table view displays all the cells of your design in a table format.

To display the table view,

- ➤ Select the *Table View* tab or, from the menu bar, choose *View – Parts Table*.
- ➤ To display instances of the selected cell, from the menu bar, choose *View – Instance Table*.

Cell Bindings Table in the Table View Tab



Library	Cell	View Found	View To Use	Inherited View List
NONE	OpAmp	**NONE**		spectre cmos_sch ...
Two_Stage_Opamp	OpAmp_AC_top	schematic		spectre cmos_sch ...
Two_Stage_Opamp	opamp_dc_feedback	schematic	schematic	spectre cmos_sch ...
analogLib	cap	spectre		spectre cmos_sch ...
analogLib	idc	spectre		spectre cmos_sch ...
analogLib	res	spectre		spectre cmos_sch ...
analogLib	vcvs			spectre cmos_sch ...
analogLib	vdc			spectre cmos_sch ...
gpdk447	ndio			spectre cmos_sch ...

The *Cell Bindings* table displays the library and view bindings of cells. The *View Found* column displays the view that is found for the cell based on the binding rules. The *View to Use* column lets you specify a specific view for the cell.

You can also edit the *Inherited View List* and *Inherited Lib List* columns. For more information about all the columns that can be displayed in the Cell Bindings table, see the description of the Options form.

You can sort the cells in the *Cell Bindings* table by clicking on any of the column headings. You can also move the columns by clicking-and-dragging the column heading.

Mark As External HDL Text (AMS UNL only)

This option is used to bind a cellview to an external text view and is applicable only in AMS UNL flow.

This option has been added as another HED cellview binding property selection for use in the AMS UNL flow that can be used to bind a cellview to an external text file, which can be accessed or passed through `-v/-y/-f/-reflib` on the `xrun` argument.

Note: The `irun` argument is also valid but will be removed in the future releases.

For more information on the AMS UNL flow, refer to the AMS Unified Netlister section of the *Virtuoso ADE Explorer User Guide*.

Cell Bindings Table Context-Sensitive Menu

Right-clicking over the contents of the *Cell Bindings* table will display a context menu that contains options specific for use with the cell bindings listed.

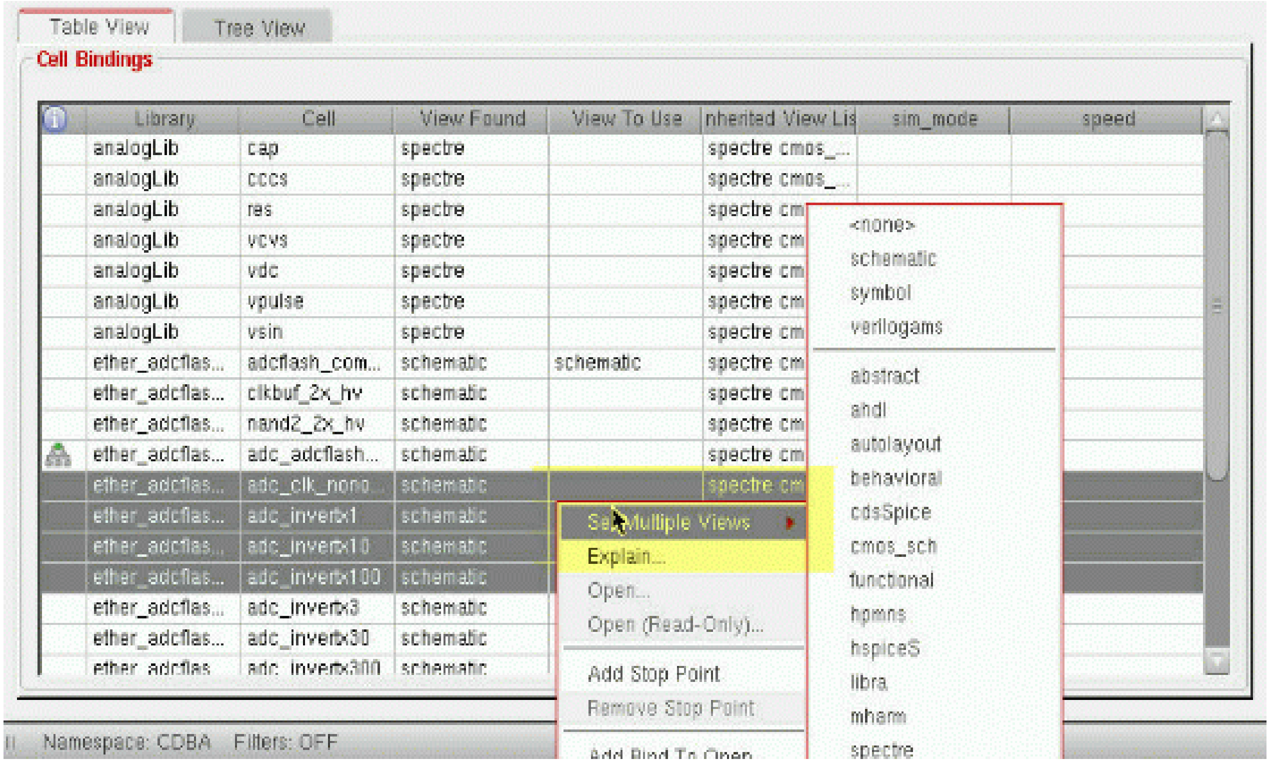
For more information on the context-sensitive menu options available, see:

- Defining Stop Points on a Per-Cell Basis (for information on adding and removing stop points)
- Verifying Cell Bindings Rules (to *Explain* which binding rules were used for a cell).
- About Sub-Configurations (for information on setting cell views)
- Defining Bind-to-Open on a Per-Cell Basis (for information on adding and removing bind to open).

Multiple Selection and Action

You can select multiple items in the *Cell Bindings* table, and apply specific operations to all of them, such as setting a binding, applying property values, setting a view list, setting stop points, and so on, using the context-sensitive menu.

Note: The Hierarchy Editor will only apply the selected operation on those items where it makes sense to do so. For example, if an item is not editable it would be skipped.



	Library	Cell	View Found	View To Use	nherited View Lis	sim_mode	speed
	analogLib	cap	spectre		spectre cmos_...		
	analogLib	cccs	spectre		spectre cmos_...		
	analogLib	res	spectre		spectre cm		
	analogLib	vcvs	spectre		spectre cm		
	analogLib	vdc	spectre		spectre cm		
	analogLib	vpulse	spectre		spectre cm		
	analogLib	vsin	spectre		spectre cm		
	ether_adcflas...	adcflash_com...	schematic	schematic	spectre cm		
	ether_adcflas...	clkbuf_2x_hv	schematic		spectre cm		
	ether_adcflas...	nand2_2x_hv	schematic		spectre cm		
	ether_adcflas...	adc_adcflash...	schematic		spectre cm		
	ether_adcflas...	adc_clk_nono...	schematic		spectre cm		
	ether_adcflas...	adc_invertx1	schematic				
	ether_adcflas...	adc_invertx10	schematic				
	ether_adcflas...	adc_invertx100	schematic				
	ether_adcflas...	adc_invertx3	schematic				
	ether_adcflas...	adc_invertx30	schematic				
	ether_adcflas	adc_invertx300	schematic				

Namespace: CDBA Filters: OFF

The following are examples of how to perform a multiple selection and apply a chosen action:

1. Select the multiple items in the *Cell Bindings* table using `Shift` or `Ctrl` with the left-mouse button.

2. Right-click over the *View to Use* column to display the context-sensitive menu.

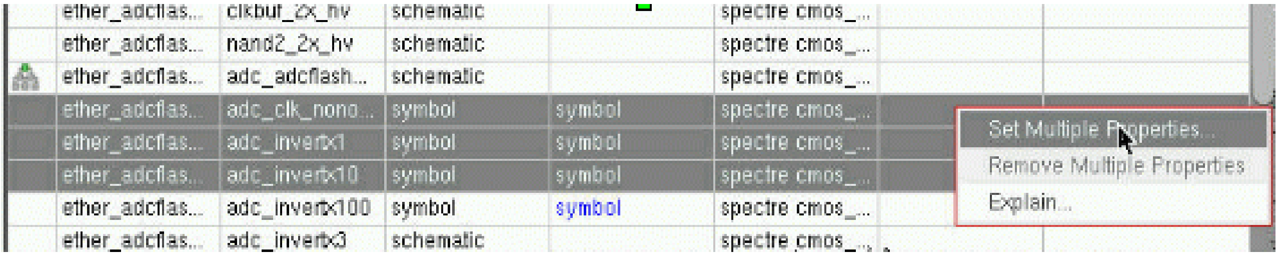
   Note: The *Set Cell View* option has been replaced in this context with *Set Multiple Views*.

3. Select *Set Multiple Views*.

4. Select the required view to be used for the multiple items selected.

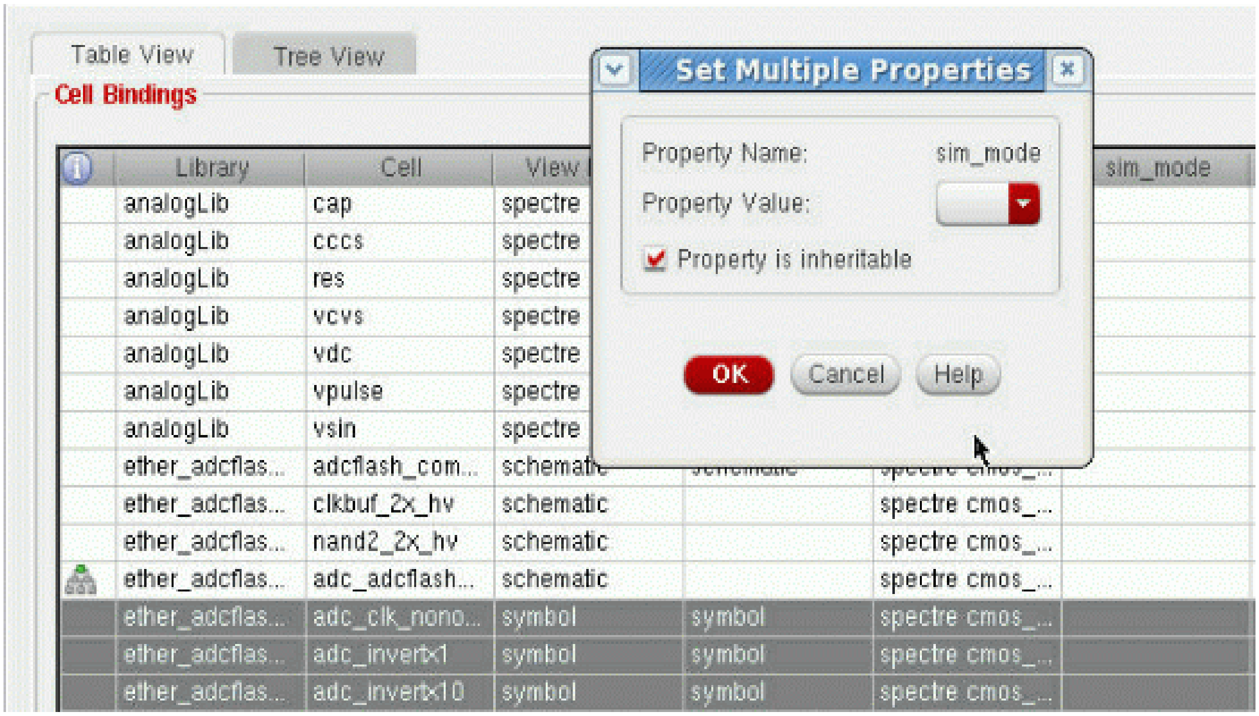
You can also apply the same technique to, for example, set a property value to a multiple selection of items:

1. Select *View – Properties* to display property information in the *Cell Bindings* table.
2. Select multiple items in the *Cell Bindings* table using `Shift` or `Ctrl` with the left-mouse button.
3. Right-click over the property column (in this example *sim_mode*) to display the appropriate context-sensitive menu.
4. Select *Set Multiple Properties*.



ether_adcflas...	clkbuf_2x_hv	schematic		spectre cmos_...
ether_adcflas...	nand2_2x_hv	schematic		spectre cmos_...
ether_adcflas...	adc_adcflash...	schematic		spectre cmos_...
ether_adcflas...	adc_clk_nono...	symbol	symbol	spectre cmos_...
ether_adcflas...	adc_invertx1	symbol	symbol	spectre cmos_...
ether_adcflas...	adc_invertx10	symbol	symbol	spectre cmos_...
ether_adcflas...	adc_invertx100	symbol	symbol	spectre cmos_...
ether_adcflas...	adc_invertx3	schematic		spectre cmos_...

The Set Multiple Properties form is displayed.



Property Name: sim_mode

Property Value:

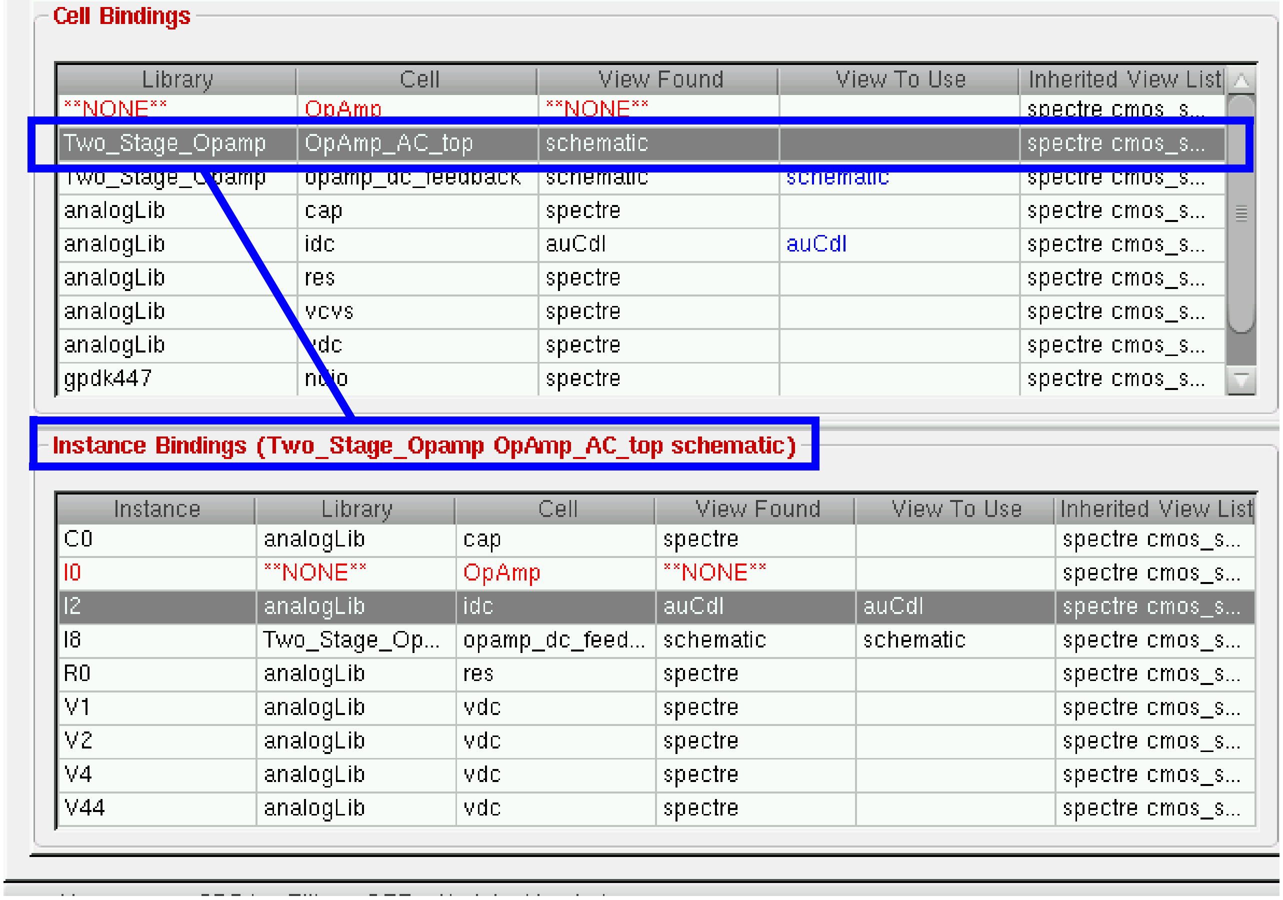
☑ Property is inheritable

Library	Cell	View ...			sim_mode
analogLib	cap	spectre			
analogLib	cccs	spectre			
analogLib	res	spectre			
analogLib	vcvs	spectre			
analogLib	vdc	spectre			
analogLib	vpulse	spectre			
analogLib	vsin	spectre			
ether_adcflas...	adcflash_com...	schematic	schematic	spectre cmos_...	
ether_adcflas...	clkbuf_2x_hv	schematic		spectre cmos_...	
ether_adcflas...	nand2_2x_hv	schematic		spectre cmos_...	
ether_adcflas...	adc_adcflash...	schematic		spectre cmos_...	
ether_adcflas...	adc_clk_nono...	symbol	symbol	spectre cmos_...	
ether_adcflas...	adc_invertx1	symbol	symbol	spectre cmos_...	
ether_adcflas...	adc_invertx10	symbol	symbol	spectre cmos_...	

5. Complete the Set Multiple Properties form as required.
6. Click *OK* to apply the changes to all selected items.

Instance Bindings Table in the Table View Tab

To display the *Instance Bindings* table, ensure that *View – Instance Table* has been selected, and that you also have an appropriate Lib/Cell/View selected in the *Cell Bindings* table.



Cell Bindings

Library	Cell	View Found	View To Use	Inherited View List
NONE	OpAmp	**NONE**		spectre cmos_s...
Two_Stage_Opamp	OpAmp_AC_top	schematic		spectre cmos_s...
Two_Stage_Opamp	opamp_dc_feedback	schematic	schematic	spectre cmos_s...
analogLib	cap	spectre		spectre cmos_s...
analogLib	idc	auCdl	auCdl	spectre cmos_s...
analogLib	res	spectre		spectre cmos_s...
analogLib	vcvs	spectre		spectre cmos_s...
analogLib	vdc	spectre		spectre cmos_s...
gpdk447	ndio	spectre		spectre cmos_s...

Instance Bindings (Two_Stage_Opamp OpAmp_AC_top schematic)

Instance	Library	Cell	View Found	View To Use	Inherited View List
C0	analogLib	cap	spectre		spectre cmos_s...
I0	**NONE**	OpAmp	**NONE**		spectre cmos_s...
I2	analogLib	idc	auCdl	auCdl	spectre cmos_s...
I8	Two_Stage_Op...	opamp_dc_feed...	schematic	schematic	spectre cmos_s...
R0	analogLib	res	spectre		spectre cmos_s...
V1	analogLib	vdc	spectre		spectre cmos_s...
V2	analogLib	vdc	spectre		spectre cmos_s...
V4	analogLib	vdc	spectre		spectre cmos_s...
V44	analogLib	vdc	spectre		spectre cmos_s...

The *Instance Bindings* table displays all the instances in the block you selected and their bindings. Instances are described in a library-cell-view format. The *View Found* column displays the view that is found for the instance based on the binding rules. The *View to Use* column lets you specify a specific view for the instance.

You can also edit the *Inherited View List* and *Inherited Lib List* columns. For more information about all the columns that can be displayed in the Cell Bindings table, see the description of the Options form.

You can sort the instances in the *Instance Bindings* table by clicking on any of the column headings. You can also move the columns by clicking-and-dragging the column heading.

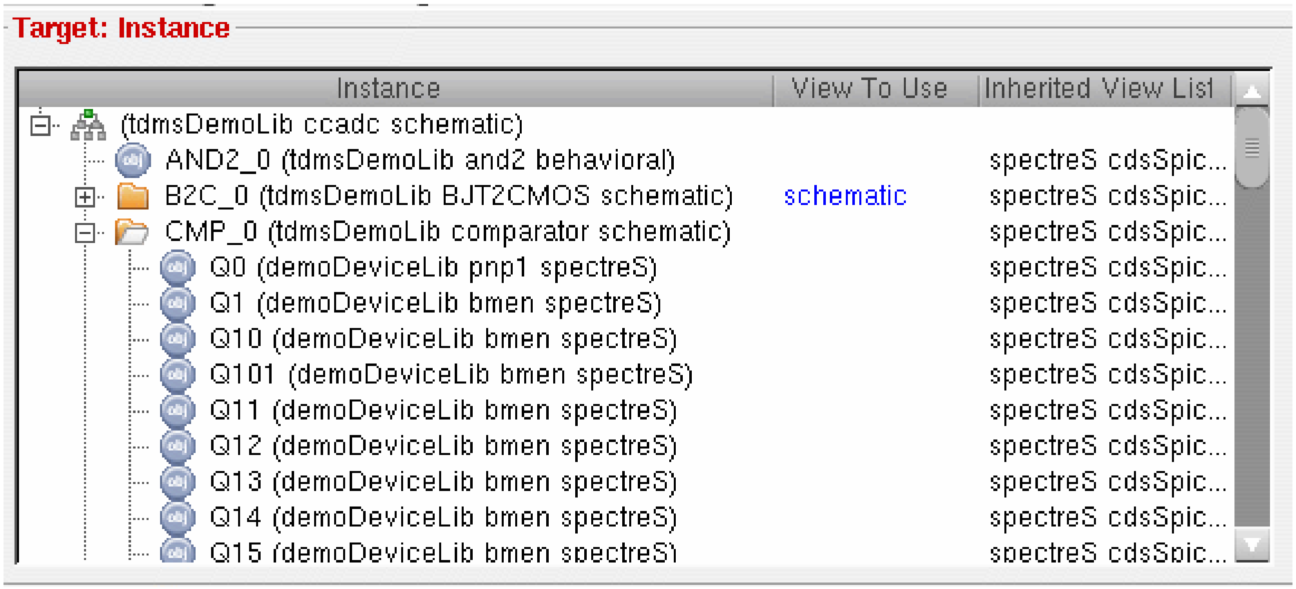
Note: Right-clicking over the *Instance Bindings* table will display the same context-sensitive menu as displayed for the *Cell Bindings* table. For more information, see Cell Bindings Table context-sensitive menu.

Tree View

The tree view displays the hierarchy of your design. It displays the top cellview and all the instances contained in it.

To display the tree view,

➤ Select the *Tree View* tab or, from the menu bar, choose *View – Tree*.



Target: Instance

Instance	View To Use	Inherited View List
(tdmsDemoLib ccadc schematic)		
AND2_0 (tdmsDemoLib and2 behavioral)		spectreS cdsSpic...
B2C_0 (tdmsDemoLib BJT2CMOS schematic)	schematic	spectreS cdsSpic...
CMP_0 (tdmsDemoLib comparator schematic)		spectreS cdsSpic...
Q0 (demoDeviceLib pnp1 spectreS)		spectreS cdsSpic...
Q1 (demoDeviceLib bmen spectreS)		spectreS cdsSpic...
Q10 (demoDeviceLib bmen spectreS)		spectreS cdsSpic...
Q101 (demoDeviceLib bmen spectreS)		spectreS cdsSpic...
Q11 (demoDeviceLib bmen spectreS)		spectreS cdsSpic...
Q12 (demoDeviceLib bmen spectreS)		spectreS cdsSpic...
Q13 (demoDeviceLib bmen spectreS)		spectreS cdsSpic...
Q14 (demoDeviceLib bmen spectreS)		spectreS cdsSpic...
Q15 (demoDeviceLib bmen spectreS)		spectreS cdsSpic...

The instance name is displayed in the **Instance** column. The library-cell-view binding of the instance is listed in parenthesis next to the name.

View to Use lets you specify a specific view for the instance.

Inherited View List displays the inherited view list that determines the view binding of the instance, unless it is overriden by a View to Use value. You can edit this column.

For more information about all the columns that can be displayed in the Cell Bindings table, see the description of the Options form.

The tree view can be set to instance mode or occurrence mode. When in occurrence mode, any bindings, properties, and stop points that you set will be set on the occurrence. When in instance mode, these commands apply to the instance.

Note: You can expand `verilog`, `verilogAMS`, `verilogA`, `systemVerilog`, `VHDL`, and `VHDLAMS` from the *Tree view* tab.

The default is instance mode; to set the occurrence mode, select the following icon on the toolbar:

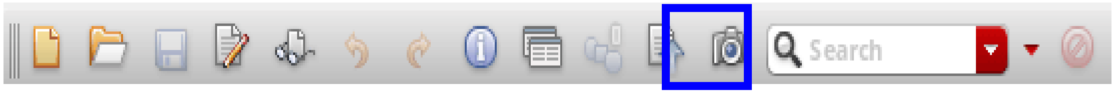
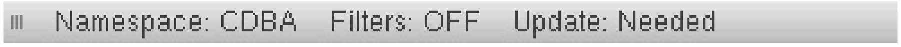


Figure 1-1 Occurrence Mode Icon

Status Bar

The status bar, whose default location is at the bottom of the main window, displays the state of the Hierarchy Editor—the name space in which you are operating, whether filters are on or off, and whether an update is required.



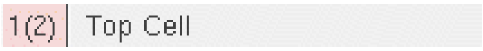
Namespace: CDBA Filters: OFF Update: Needed

You can move the status bar to any side of the main window. You cannot float it.

To move the status bar,

➤ Drag it by its handle and place it on any inside edge of the main window.

Window Border



1(2) Top Cell

The window border displays the window number. For example, 1(2), where 1 is the session window number and 2 is the window number of the Hierarchy Editor.

The window border also displays the command that is currently selected or that was last selected.

Customizing Hierarchy Editor Columns

The Hierarchy Editor lets you specify the columns you want to display, change the column order, and resize the columns.

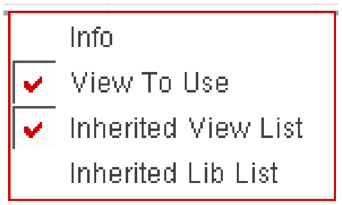
Specifying the Columns to Display

You can select the columns you want to display from the Options form or from a pop-up menu.

To select columns from the pop-up menu,

1. Right-click the column heading.

   For example, the following pop-up menu appears when right-clicking over the *Cell Bindings* table.

   Info
   ✓ View To Use
   ✓ Inherited View List
   Inherited Lib List

   The pop-up menu lists all the columns that can be displayed. Property columns appear in the list only if the *View – Properties* option is selected. Columns that are currently displayed have a check-mark next to them.

2. Select the columns that you want to display and deselect the ones that you want to hide.

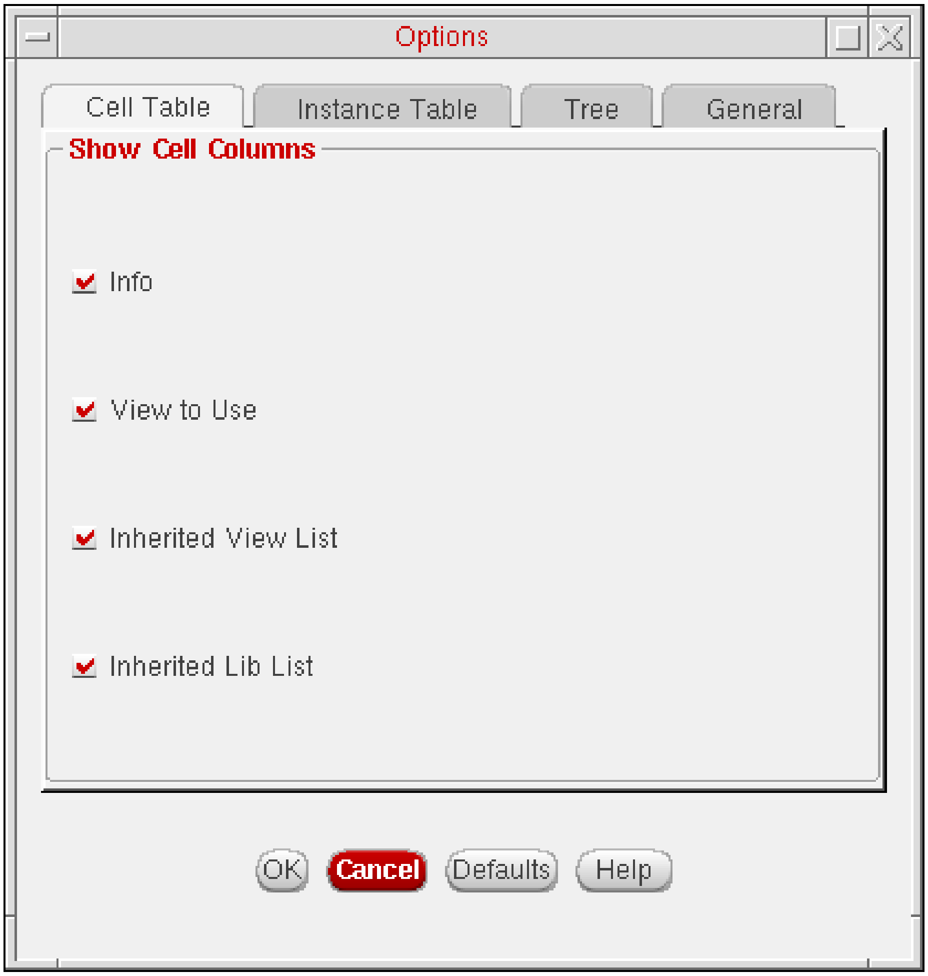
   The tree view or table view display is updated to reflect your choices. These settings remain in effect for the session only unless you save your defaults.

   Note: The display/hide setting of individual property columns is not saved in the defaults file.

To specify columns in the Options form,

1. From the menu bar, choose *View – Options*.

The Options form appears.



☑ Info

☑ View to Use

☑ Inherited View List

☑ Inherited Lib List

2. Click a tab to display a specific set of related attributes.

   For example, to select the attributes you want to display in the tree structure, click the *Tree* tab.

3. Select the attributes you want to display. To select the default attributes, click *Defaults*.

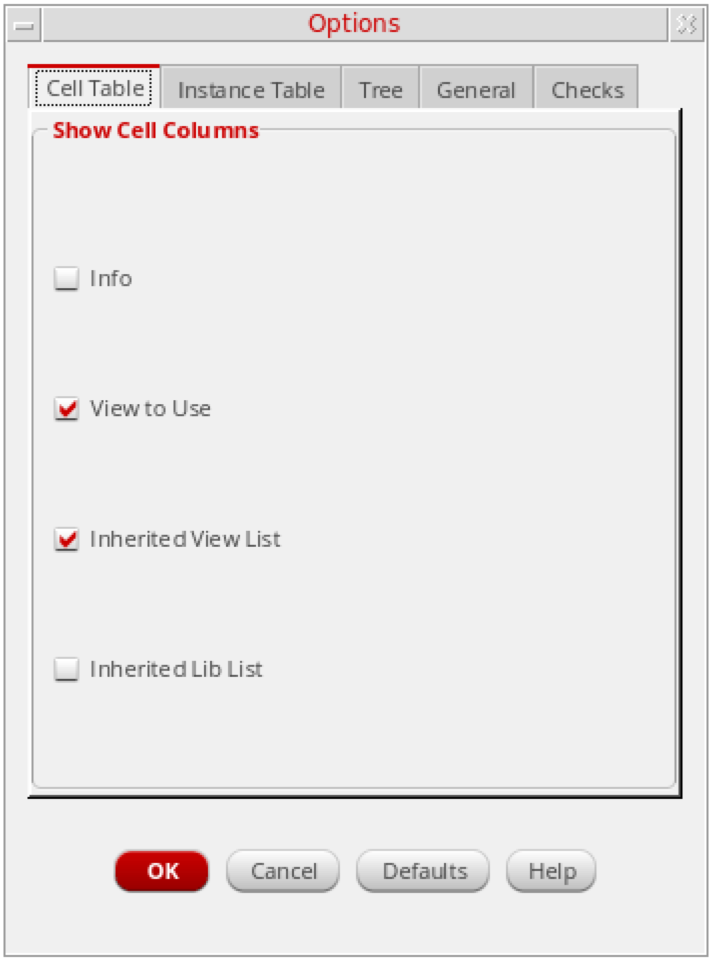
4. Click *OK* to apply your changes and close the form.

About the Options Form

The Options form lets you specify the attributes you want to display in the cell table*,* instance table*,* and tree view of the Hierarchy Editor. It also lets you select the automatic update option.

The **Cell Table tab** lets you select the columns that appear in the cell table.

Show Cell Columns



Show Cell Columns

☐ Info

☑ View to Use

☑ Inherited View List

☐ Inherited Lib List

Info displays information about the cellview.

For example, a green pyramid in this column indicates the top-level cell view. For a complete list of icons that can be displayed in this column and their meanings, choose *Help – Legend* to display the Legend dialog box.

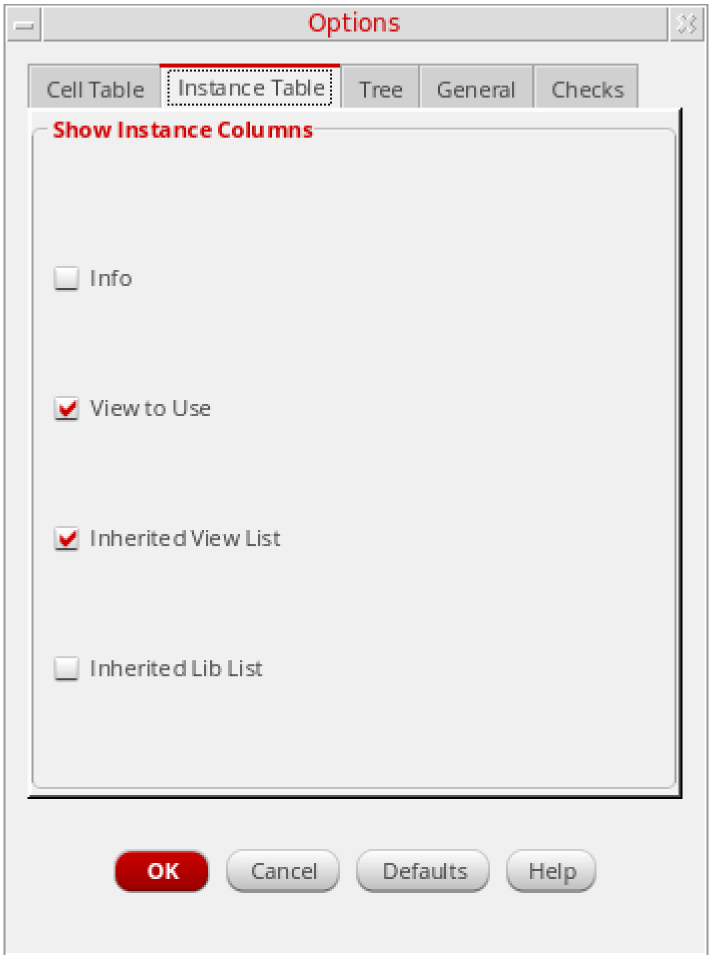
View to Use displays a field in which you can type a specific view to use for the cell.

Inherited View List displays a field in which you can modify the inherited view list for the cell. The default inherited view list is the global view list.

Inherited Lib List displays a field in which you can modify the inherited library list for the cell. The default inherited library list is the global library list.

The **Instance Table tab** lets you select the columns that appear in the instance table.

Show Instance Columns



□ Info

☑ View to Use

☑ Inherited View List

□ Inherited Lib List

Info displays information about the instance.

For example, a green pyramid in this column indicates the top-level cell view. For a complete list of icons that can be displayed in this column and their meanings, choose *Help – Legend* to display the Legend dialog box.

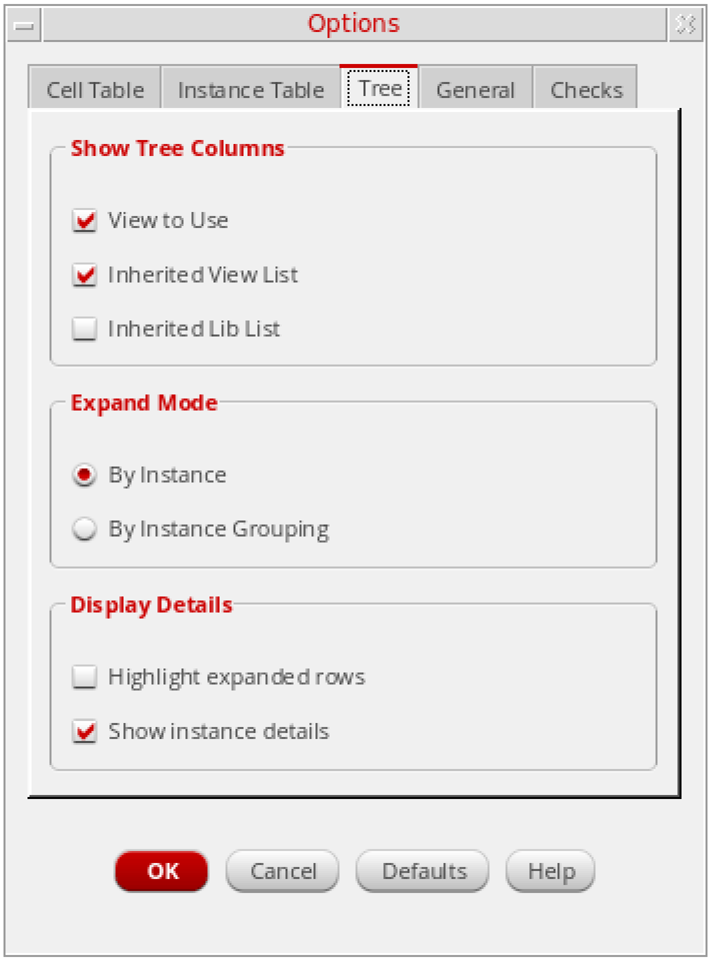
View to Use displays a field in which you can type a specific view to use for the instance.

Inherited View List displays a field in which you can modify the inherited view list for the instance.

Inherited Lib List displays a field in which you can modify the inherited library list for the instance.

The **Tree tab** lets you select the columns that appear in the tree structure.

Show Tree Columns



Show Tree Columns

☑ View to Use

☑ Inherited View List

□ Inherited Lib List

Expand Mode

● By Instance

○ By Instance Grouping

Display Details

□ Highlight expanded rows

☑ Show instance details

View to Use displays a field in which you can type a specific view to use for the instance or occurrence.

Inherited View List displays a field in which you can modify the inherited view list for the instance or occurrence.

Inherited Lib List displays a field in which you can modify the inherited library list for the instance or occurrence.

Expand Mode

By Instance displays all the instances for the selected cell when you expand a tree node.

By Instance Grouping displays similar instances grouped together when you expand a tree node. The number of instances is also displayed if the *Show instance details* option has been selected.

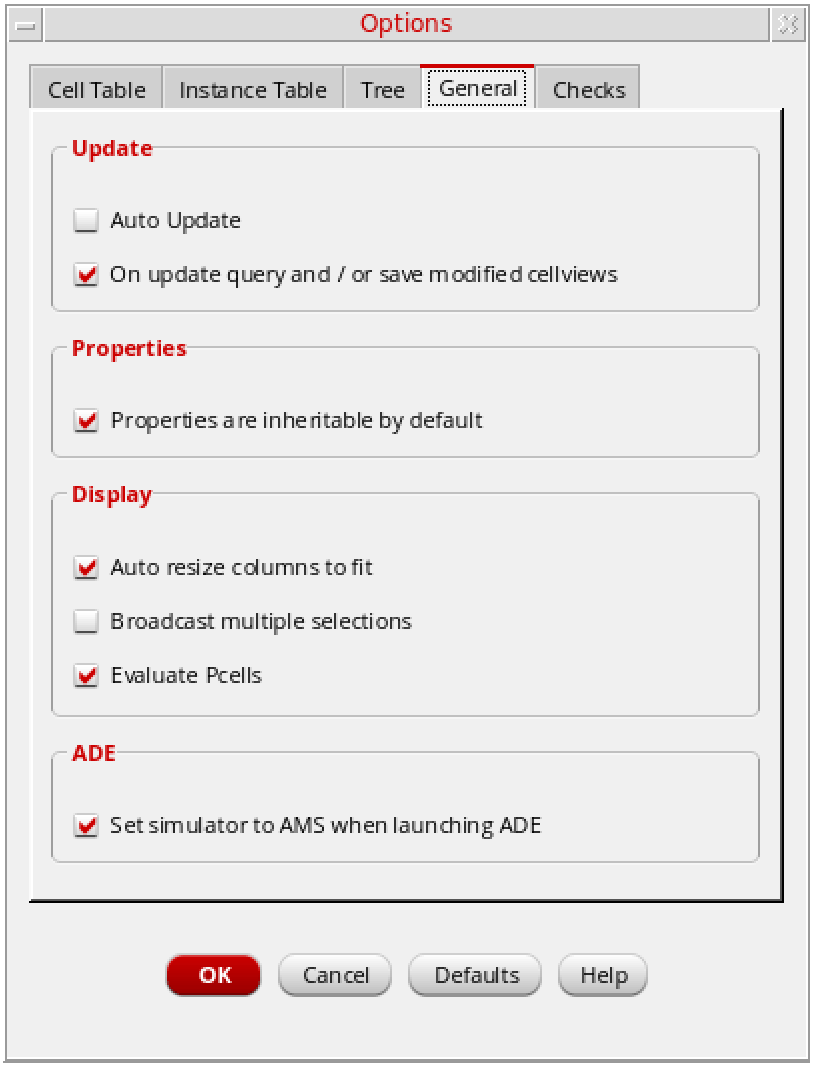
The default expand mode is *By instance*.

Display Details

Highlight expanded rows highlights instances in expanded tree nodes.

Show instance details displays the library-cell-view to which the instance is bound, in addition to the instance name. For example: *Q1(tdmsDemoLib bmen spectreS)*. It also displays the number of instances for the cell if the *Expand Mode* is set to *By Instance Grouping*.

The **General tab** lets you set the following option:



Update

□ Auto Update

☑ On update query and / or save modified cellviews

Properties

☑ Properties are inheritable by default

Display

☑ Auto resize columns to fit

□ Broadcast multiple selections

☑ Evaluate Pcells

ADE

☑ Set simulator to AMS when launching ADE

Auto Update updates the configuration automatically every time you make a change. If this option is selected, you do not have to click the *Update* icon in the toolbar or choose *View – Update* to recompute the hierarchy or view your changes.

On update query and / or save modified cellviews check this option to display the updateSync form. Unchecking this option will mean that this form is not displayed when an update is performed.

> **Note:** updateSync is only required when you want to synchronize changes to the configuration with an external process.

Properties are inheritable by default if unchecked, then set properties will not be inherited down the hierarchy.

Auto resize columns to fit if unchecked, columns will not be re-sized. Therefore, if a window is reduced in size, a scrollbar will be added. By default, columns will be auto-resized to fit the window.

Broadcast multiple selections if checked, and multiple items are selected, then all selected items will be broadcast to the Virtuoso Schematic Editor, as well as being cross-selected there. By default, only single selections are broadcast. For example, if the top cell schematic is opened "in-context", and multiple instances are selected in the Hierarchy Editor, then only the last selection will be cross-selected in the schematic.

Evaluate Pcells if checked (default), the Hierarchy Editor will perform Pcell evaluation. Choosing to enable Pcell evaluation will instruct the Hierarchy Editor to perform a more complete elaboration of the design hierarchy.

Note: Enabling this option can however have an impact on software performance.

Set simulator to AMS when launching ADE if checked (default), will set the simulator to AMS. If not selected, HED will not change the simulator and it will be based on the current ADE startup simulator .cdsenv value.

```
hed.ade setAmsSimulator   boolean   t/nil
```

- ❑ If `t` (default) is specified, the simulator value will be set to "`ams`".
- ❑ If `nil` is specified, the simulator value will not be modified by HED and will be based on the ADE startup simulator .cdsenv value.

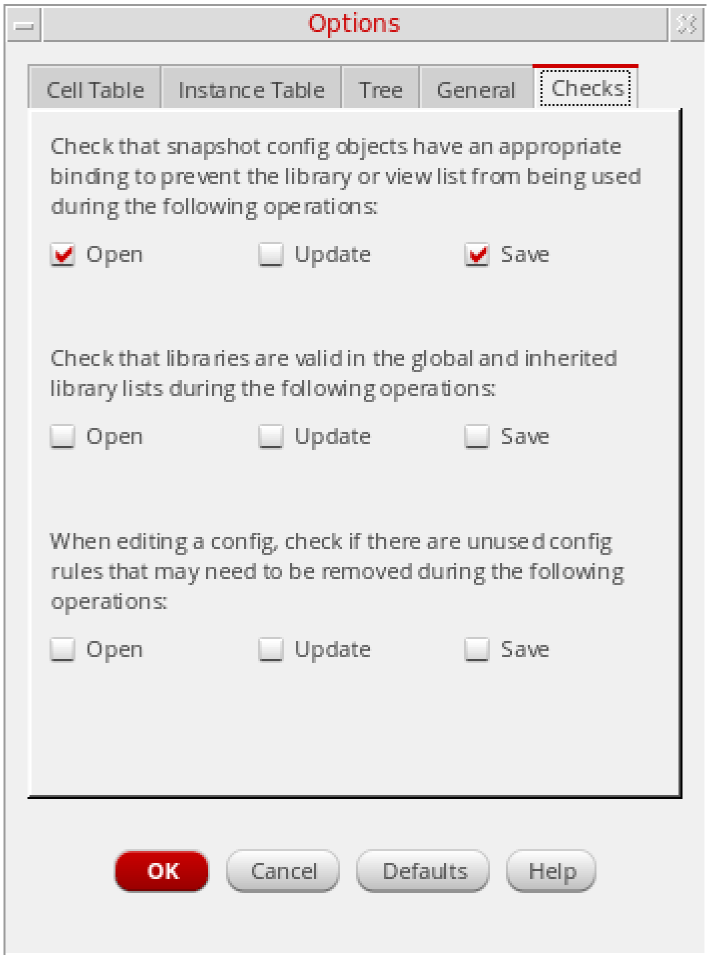
OK applies your selections and closes the form.

Cancel closes the form without applying your selections.

Defaults resets the values to their defaults.

Help opens help in this area.

The **Checks tab** lets you set the following options:



Check that snapshot config objects have an appropriate binding to prevent the library or view list from being used during the following operations:

☑ Open ☐ Update ☑ Save

Check that libraries are valid in the global and inherited library lists during the following operations:

☐ Open ☐ Update ☐ Save

When editing a config, check if there are unused config rules that may need to be removed during the following operations:

☐ Open ☐ Update ☐ Save

Snapshot config objects Checks if the snapshot config objects have an appropriate binding to prevent the library or view list from being used while opening, updating, or saving the config.

Libraries Check if libraries are valid in the global and inherited library list while opening, updating, or saving the config.

Editing a config Checks for unused config rules that need to be removed while opening, updating, or saving the config.

Changing Column Order

You can change the order of the columns in the table view or tree view. For example, you can have the *Cell* column appear before the *Library* column.

To move a column,

➤ Drag and drop the column heading to the new location.

Resizing Columns

You can change the width of any column by dragging the column heading.

To resize a column,

1. Place the cursor on the right border of the column heading till the double arrows appear.
2. Drag the border to the right to make the column wider or drag it to the left to make it narrower.

When you resize a column, the right-most column is adjusted to accommodate the new size. The other columns are not resized.

Sorting Data

You can sort data in the cell or instance table by column.

To sort by column,

1. Click the heading of the column by which you want to sort data. For example, to sort cells in the cell table by the views found, click the *View Found* column heading.

   The column is sorted in alphabetical order.

2. Click the heading again to change the alphabetical order from ascending to descending or vice versa.

Updating the Configuration Automatically

You can choose to have the configuration automatically updated whenever you make a change. For example, if you open a configuration and add a view to the global view list, the Hierarchy Editor will automatically recompute the hierarchy and display the updated bindings. If you do not set the automatic update option, you would need to click the *Update* icon or choose *View – Update* to view the changes you make.

The automatic update option is turned off by default.

To set the automatic update option,

1. From the menu bar, choose *View – Options*.

   The Options form appears.

2. Select the *General* tab.
3. Select *Auto Update*.
4. Click *OK* to apply your change and close the Options form.

Displaying the Tree View or Table View

The tree view displays the hierarchy of your design while the table view displays all the cells of your design in a table format.

To display the tree view,

➤ Select the *Tree View* tab or, from the menu bar, choose *View – Tree*.

To display the table view,

➤ Select the *Table View* tab or, from the menu bar, choose *View – Parts Table*.

➤ To display instances of the selected cell, from the menu bar, choose *View – Instance Table*.

See also Table View and Tree View.

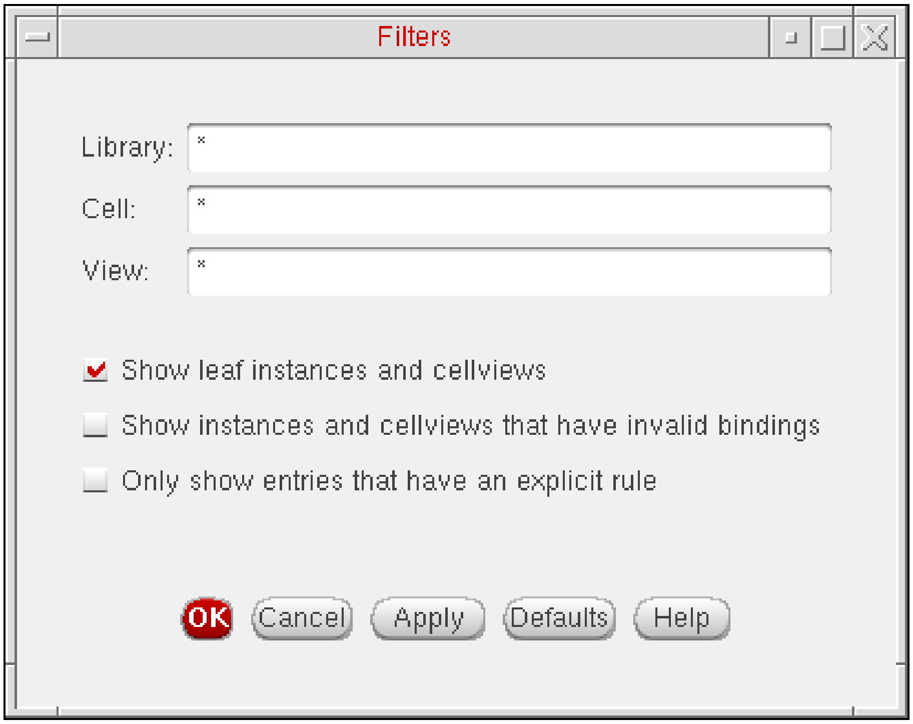
Filtering Cellviews

You can find cellviews quickly by using filters. You can set filters to display only those cellviews that match a certain pattern or those that belong to a particular library, for example. You can also choose to filter out cellviews that are at the leaf level, have an invalid binding, or do not have an explicit rule.

To filter cellviews,

1. Choose *View – Filters*.

   The Filters form appears.



Library: *

Cell: *

View: *

☑ Show leaf instances and cellviews

□ Show instances and cellviews that have invalid bindings

□ Only show entries that have an explicit rule

2. In the *Library*, *Cell*, and *View* fields, specify whole words or character strings to narrow your cellview search.

   You can use the following wildcards:

Character	**Match Criteria**
`?`	Matches any single character
`[`*`list`*`]`	Matches any single character in *`list`*

Character	Match Criteria
`[`*lower-upper*`]`	Matches any character in the range between *lower* and *upper*
`*`	Matches any pattern

For example, to find all the libraries that begin with `p`, in the *Library* field, type `p*`.

Pattern matching is case sensitive. You can also specify multiple patterns by separating the patterns with spaces.

3. Select the type of data you want to see by selecting one or more of the following options:

   ❑ *Show leaf instances and cellviews*

   Displays even those instances and cellviews that are at the leaf level of the tree, that is, they do not have a hierarchy under them.

   ❑ *Show instances and cellviews that have invalid bindings*

   Displays even those instances and cellviews that are unbound or whose binding is invalid.

   ❑ *Only show entries that have an explicit rule*

   Displays only the entries for which a view or view list has been set explicity in the *View to Use* or *Inherited View List* column respectively, that is, the inherited value has been overriden. Entries with an explicit Bind-to-Open, Source File, or Reference Verilog setting are also displayed when this option is selected.

4. Click *Apply*.

   The status bar displays *Filters: ON*.

5. Click *OK*.

   The Filters form closes and the Hierarchy Editor uses the selected filter criteria until you turn off the filters.

   Note: The filter setting returns to the default setting when you start another Hierarchy Editor session. You must save your settings using the *File – Save Defaults* command to save the filter settings for future sessions.

About the Filters Form

Library lets you specify the search criteria for library names.

Cell lets you specify the search criteria for cell names.

View lets you specify the search criteria for view names.

For the *Library*, *Cell*, and *View* fields, you can:

- ❑ Specify whole words or character strings
- ❑ Specify multiple patterns; separate the patterns with spaces
- ❑ Use the following wildcards:

?	Matches any single character
[*list*]	Matches any single character in *list*
[*lower-upper*]	Matches any character in the range between *lower* and *upper*
*	Matches any pattern

Note: Pattern matching is case sensitive.

Show leaf instances and cellviews lets you specify whether you want to display instances and cellviews that are at the leaf level of the tree, that is, they do not have any items under them.

Show instances and cellviews that have invalid bindings lets you specify whether you want to display instances and cellviews that are unbound or whose binding is invalid.

Only show entries that have an explicit rule lets you specify whether you want to display all entries or only those entries for which a view or view list has been set explicitly in the *View to Use* or *Inherited View List* column respectively (that is, the inherited values have been overriden). Entries with an explicit Bind-to-Open, Source File, or Reference Verilog setting are also displayed when this option is selected.

OK applies your settings and closes the form.

Cancel closes the form without saving any settings.

Apply applies your settings and leaves the form open.

Defaults restores the default match criteria and turns the filter off. The Hierarchy Editor displays all cellviews and instances. The status bar displays *Filters OFF*.

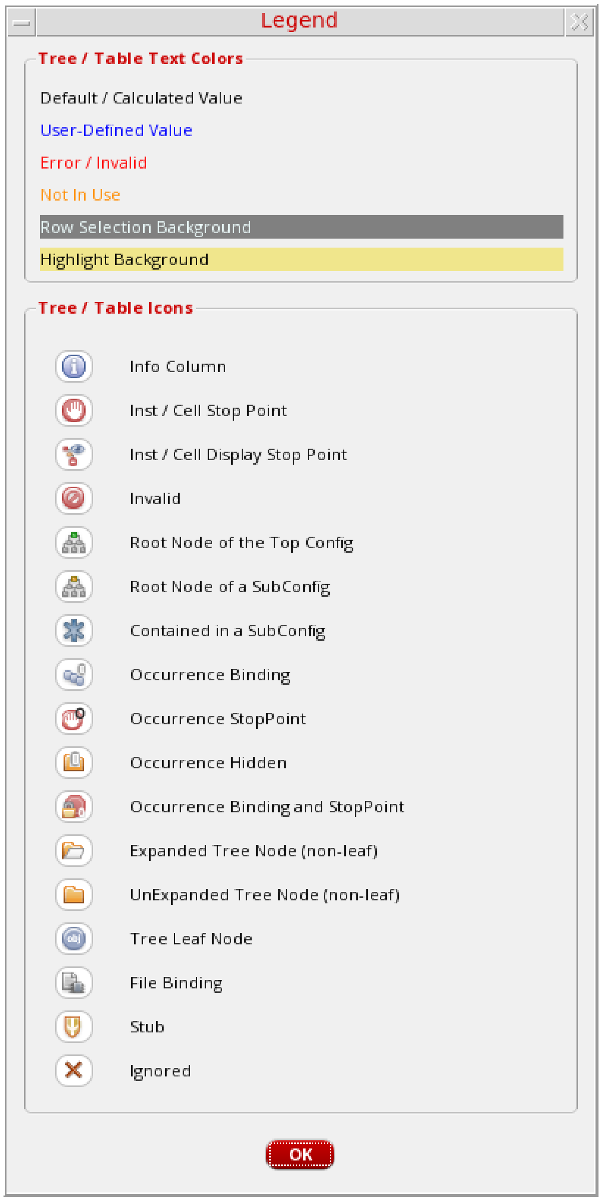
Help opens this manual.

Viewing the Legend

The Legend dialog box provides information about the icons and colors that are used in the Hierarchy Editor user interface.

To display the Legend dialog box,

➤ From the menu bar, choose *Help – Legend*.



Tree / Table Text Colors

- Default / Calculated Value
- User-Defined Value
- Error / Invalid
- Not In Use
- Row Selection Background
- Highlight Background

Tree / Table Icons

- Info Column
- Inst / Cell Stop Point
- Inst / Cell Display Stop Point
- Invalid
- Root Node of the Top Config
- Root Node of a SubConfig
- Contained in a SubConfig
- Occurrence Binding
- Occurrence StopPoint
- Occurrence Hidden
- Occurrence Binding and StopPoint
- Expanded Tree Node (non-leaf)
- UnExpanded Tree Node (non-leaf)
- Tree Leaf Node
- File Binding
- Stub
- Ignored

You can customize the text colors. For more information, see “Changing Binding Data Color Definitions” on page 194.

Changing the Hierarchy Editor Fonts

To change the Hierarchy Editor fonts,

1. Create a file named `.hedinit` (for standalone HED) or `.cdsinit` (for Virtuoso HED) in your home directory or current working directory.
2. In the `.hedinit` or `.cdsinit` file, add an `hiSetFont` SKILL function call specifying the font to use.

   Note: The Hierarchy Editor only supports the `label` font type for the *`t_fontType`* argument.

   For example:

   ```
   hiSetFont("text" ?name "Roboto Mono" ?size 12)

   hiSetFont("ciw" ?name "Roboto Mono" ?size 12)

   hiSetFont("label" ?name "Open Sans" ?size 12)
   ```

   For more information about `hiSetFont`, see the *Cadence User Interface SKILL Functions Reference*.
3. Restart the Hierarchy Editor.

You can have multiple `.hedinit` or `.cdsinit` files. The Hierarchy Editor looks for the following files, in this order, and uses the first file that is found:

- *`your_install_dir`*`/tools/dfII/local/.hedinit`

  Use this directory if you want to customize fonts for your entire site.

  Note: If you create a site `.hedinit` or `.cdsinit` file, you should include the following in the file:

  ```
  if( and( isFile( "./.hedinit" ) isReadable( "./.hedinit" ) )
     then loadi( "./.hedinit" )
     else when( and( isFile( "~/.hedinit" ) isReadable( "~/.hedinit" ) )
               loadi( "~/.hedinit" )
          )
  )
  ```

  This loads users' `./.hedinit` or if it exists, `~/.hedinit` if that exists, and is needed because the Hierarchy Editor only reads the first `.hedinit` file that is found.

  ❑ `./.hedinit`

  ❑ `~/.hedinit`

- *`your_install_dir`*`/tools/dfII/local/.cdsinit`

```
if( and( isFile( "./.cdsinit" ) isReadable( "./.cdsinit" ) )
   then loadi( "./.cdsinit" )
   else when( and( isFile( "~/.cdsinit" ) isReadable( "~/.cdsinit" ) )
             loadi( "~/.cdsinit" )
        )
)
```

This loads users’ `./.cdsinit` if it exists, or `~/.cdsinit`, if that exists, and is needed because the Hierarchy Editor only reads the first `.cdsinit` file that is found.

❑ `./.cdsinit`

❑ `~/.cdsinit`

Saving Custom Settings

When you exit the Hierarchy Editor, your settings are not automatically saved. You can save your settings with the Save Defaults form.

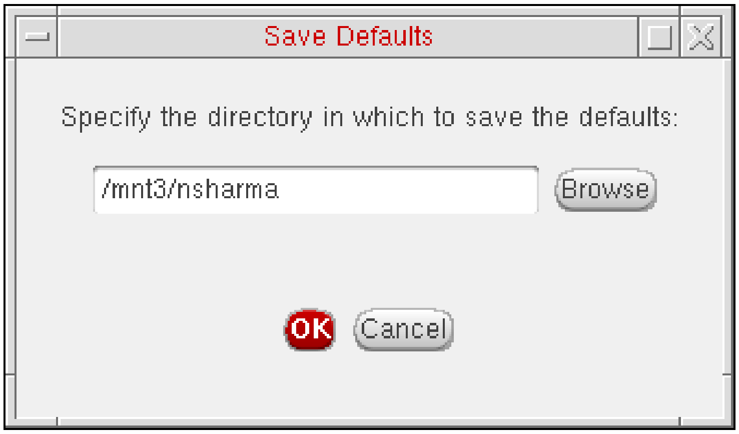
You can save and restore the following settings:

- The height and width of the Hierarchy Editor window
- The location of the Hierarchy Editor window on the screen
- The filter settings
- The view options to show or hide the Top Cell window, Global Bindings window, the tree view or table view, Instance Bindings table, and properties
- The size of the instance table, tree structure, and message area
- The columns that are displayed, including property columns
- The Hierarchy Editor form values
- Plug-in settings

To save your current settings,

1. Choose *File – Save Defaults*.

   The Save Defaults form appears:



Save Defaults

Specify the directory in which to save the defaults:

/mnt3/nsharma

2. Type the path to the directory in which you want to save the settings.
3. Click *OK*.

   The information is saved in the `hed.env` file in the directory you specified.

The next time you start the Hierarchy Editor, it loads the `hed.env` file. If the `hed.env` file is not in a directory that is found by the Cadence search mechanism, you can load the file by starting the Hierarchy Editor with the `-restore` *`dirname`* command. For more information about the `hed.env` file, see Appendix A, "The hed.env File."

Exiting the Hierarchy Editor

To close the Hierarchy Editor,

1. Choose *File – Exit* or press `Control-q`.

   The Hierarchy Editor closes.

   If you have not saved the configuration changes you have made, a *Warning* form asks you whether you want to save your changes.

2. On the *Warning* form, click one of the following:

   ❑ *Yes* to save your changes and exit

   ❑ *No* to discard your changes and exit

   ❑ *Cancel* to dismiss the form without exiting

Note: The Exit command closes only the Hierarchy Editor. It does not close other applications with which the Hierarchy Editor might have been communicating.

2

Creating Configurations

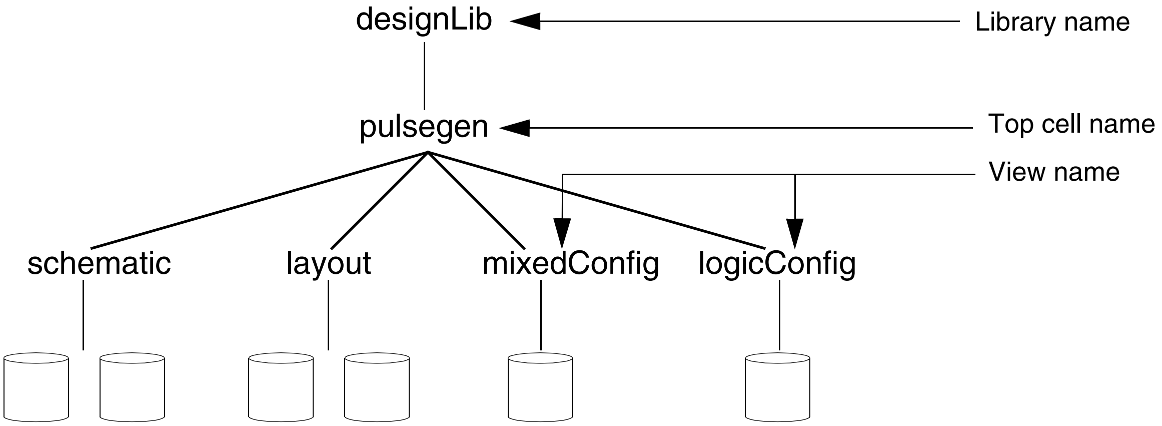
This chapter covers the following topics:

About the Configuration View of a Design Hierarchy

The configuration view of a design is the definition that the Hierarchy Editor understands and can open. It is a cellview that stores a configuration file that contains the libraries, cells, and views of a design. You can browse

- The library directory, which contains a collection of cells that correspond to a specific process technology
- The cell directory, which contains the design object that forms an individual building block of a chip or system
- The view, which is a defined representation of a cell such as layout or a schematic

The main components of a configuration are the top cellview, which is the root of the design and the configuration rules.



In the above example, the `designLib` library contains two configuration definitions identified by the view names `mixedConfig` and `logicConfig`. You can use the Hierarchy Editor to open the `mixedConfig` or the `logicConfig` configuration.

Opening Configurations

To open an existing configuration from the command-line when you start the Hierarchy Editor,

➤ In a terminal window, type

```
cdsHierEditor -lib libname -cell cellname -view viewname
```

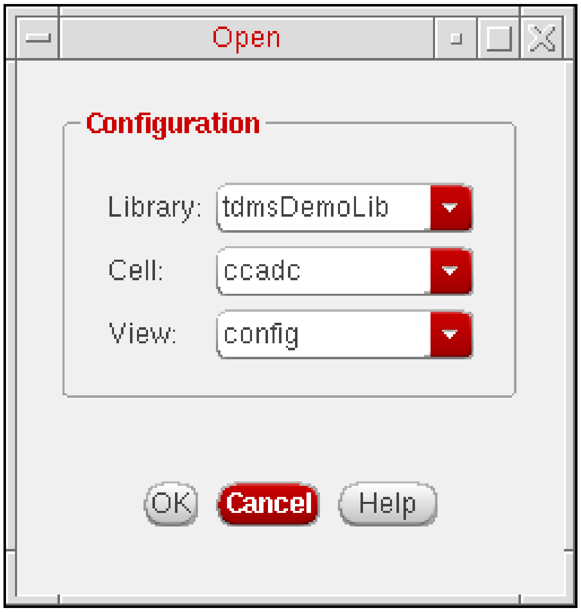
You must specify all of the above options for the Hierarchy Editor to open a configuration and load the data. If you do not specify a cellview (that is, the library, cell, and view names), the Hierarchy Editor opens without loading a configuration.

Note: If the Hierarchy Editor cannot open the configuration for editing—for example, if the configuration does not have edit permissions or if it is already locked by another process—it brings up a dialog box that asks if you want to open the configuration in read-only mode. Click *Yes* to open the configuration in read-only mode.

To open a configuration from the Hierarchy Editor,

1. From the menu bar, choose *File – Open* or press `Control-o`.

   The Open form appears.



Library: tdmsDemoLib
Cell: ccadc
View: config

2. In the *Library* field, select or type the name of the library that contains the design.

   The drop-down list contains all the libraries that are defined in your library definition file (`cds.lib`) file. For more information about library definition files, see the *Cadence Application Infrastructure User Guide*.

3. In the *Cell* field, select or type the name of the cell that contains the configuration view.

   The drop-down list contains all the cells in the library you selected.

4. In the *View* field, select or type the name of the configuration view.

   The drop-down list only displays configuration views of the cell that you selected; it does not display any other views.

5. Click *OK*.

   The Hierarchy Editor opens the configuration.

   You can use the `maskLayoutStopLimit` variable in the `hed.env` file to specify a size limit for maskLayout cellview databases. If a configuration includes such databases that are larger than the specified size limit, they will not be opened and will be shown as leaves with display stop point icons. These display stop points will not affect netlisting or simulation. For more information about the `hed.env` file, see Appendix A, "The hed.env File."

   Note: If the Hierarchy Editor cannot open the configuration for editing—for example, if the configuration does not have edit permissions or if it is already locked by another process—it brings up a dialog box that asks if you want to open the configuration in read-only mode. Click *Yes* to open the configuration in read-only mode.

Opening Configurations in Read-Only Mode

To open a configuration in read-only mode from the Hierarchy Editor,

1. From the menu bar, choose *File – Open (Read-Only)*.

   The Open (Read-Only) form appears.

2. Refer to the steps in "Opening Configurations" on page 83.

About the Open Form

Configuration

Library lets you type or select the name of the library that contains the design. The drop-down list displays all the libraries that are specified in your `cds.lib` file.

Cell lets you type or select the name of the cell that contains the configuration view. The drop-down list displays all the cells in the library you selected.

View lets you type or select the name of the configuration view you want to open. The drop-down list contains all the configuration views that are in the cell you selected.

OK opens the configuration you selected and closes the form.

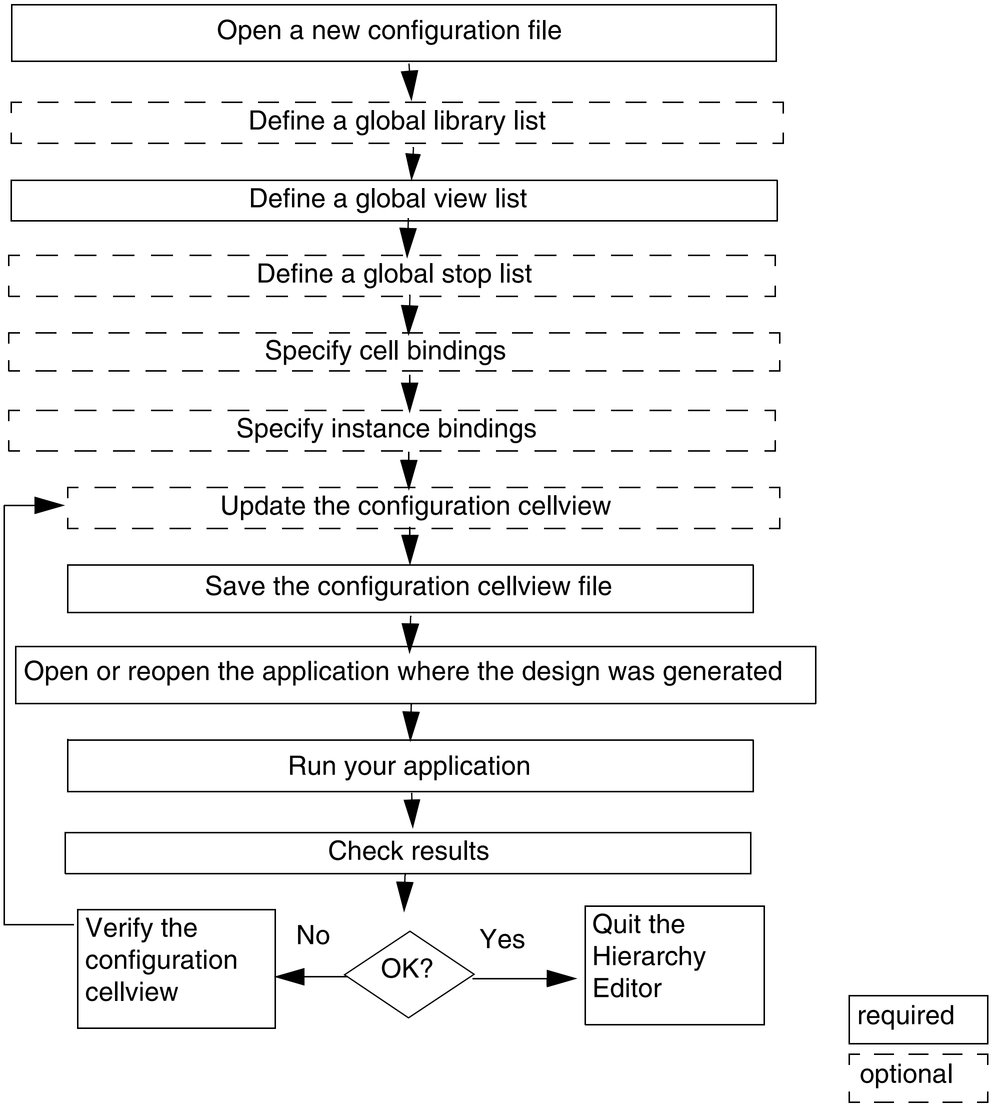
If the Hierarchy Editor cannot open the configuration for editing—for example, if the configuration does not have edit permissions or if it is already locked by another process—it brings up a dialog box that asks if you want to open the configuration in read-only mode. Click *Yes* to open the configuration in read-only mode.

Cancel cancels your selections and closes the form.

Help opens this manual.

Creating a New Configuration

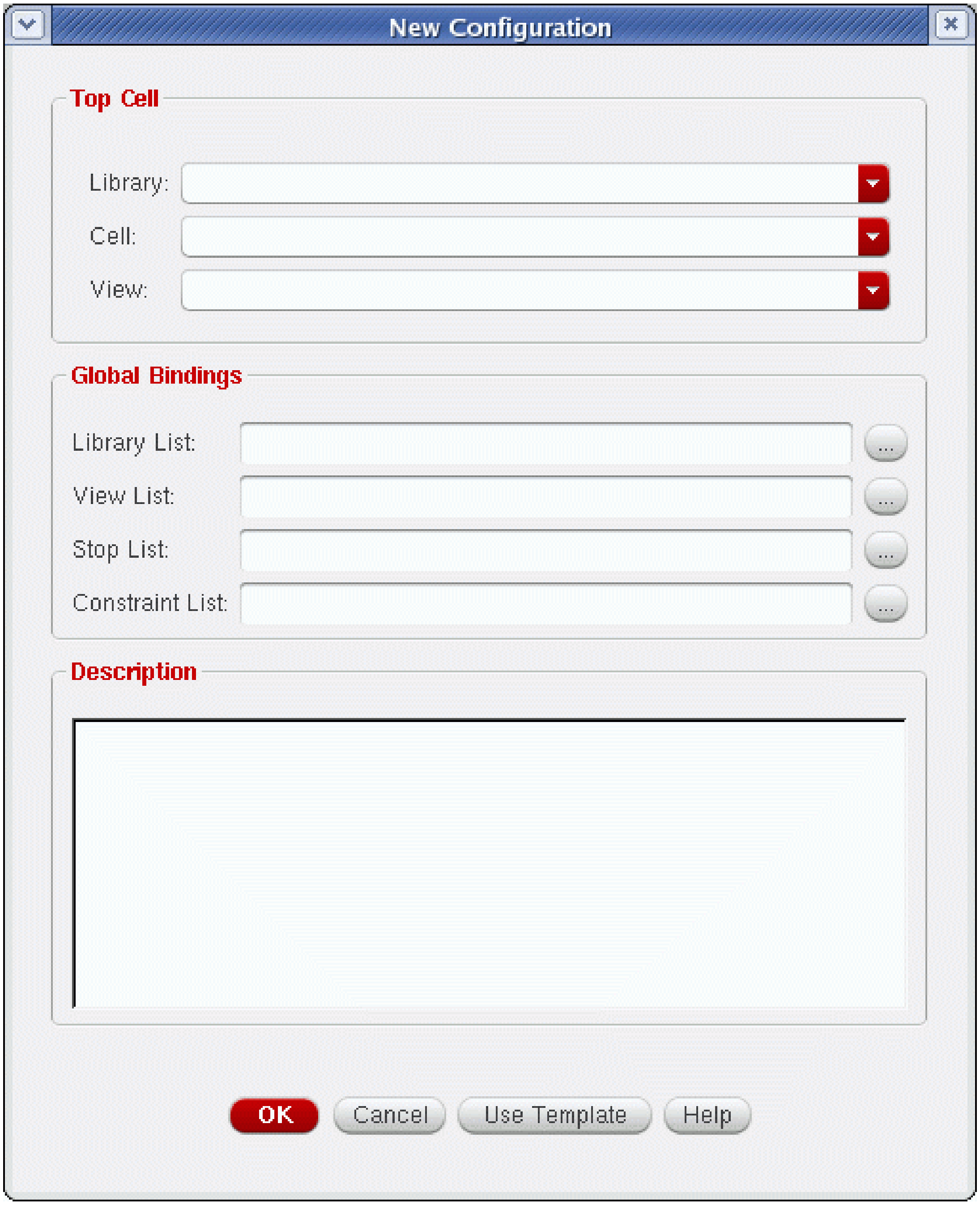
Use these steps to create and use a configuration with the Hierarchy Editor:


Open a new configuration file
Define a global library list
Define a global view list
Define a global stop list
Specify cell bindings
Specify instance bindings
Update the configuration cellview
Save the configuration cellview file
Open or reopen the application where the design was generated
Run your application
Check results
OK?
No
Yes
Verify the configuration cellview
Quit the Hierarchy Editor
required
optional

To create a new configuration,

1. Choose *File – New Config* or press `Control-n`.

The New Configuration form appears.



2. In the *Top Cell* section, in the *Library*, *Cell*, and *View* fields, type or select the name of the library, cell, and view that you want to use as the top cellview of your design.

3. In the *Global Bindings* section, do one of the following:

- ❑ In the *Library List, View List, Stop List*, and *Constraint List* fields, specify the default bindings for the entire design (see also Global Bindings Section on page 46)

  The global library list is a list of libraries that determines the libraries from which each cell is obtained. The global view list is a list of views that determines which view is selected for each object in the design. The global stop list specifies a list of views that are to be treated as leaf nodes, that is, they are not to be expanded. The global constraint list is a list of constraint views that determines the constraints that apply to the design.

  The *Stop List* and *Constraint List* fields are optional.

  List the entries in each list in order of preference. Separate entries with spaces.

  You can use constants in the view list and stop list. A constant is a symbolic name used to represent a set of views. For information on creating and using constants, see "Using Constants" on page 189.

  You can use the asterisk character (*) as a wildcard in the view list; see "Using Wildcards in a View List" on page 193 for more information.

- ❑ Click *Use Template* to select a template that is compatible with the simulator you are running. Templates for simulators provide lists of views that are most often used for those simulators.

  Refer to "Working with Templates" on page 106 to use predefined templates or to create your own templates.

  Note: If the template you selected has filled in the *Top Cell* section of the New Configuration form, replace the names in the *Library*, *Cell*, and *View* fields with the top cellview you want to use.

4. (Optional) In the *Description* field, type a brief statement describing the new configuration.
5. Click *OK*.

   The New Configuration form closes. The Hierarchy Editor displays the new configuration.

See "Saving Configurations" on page 111 for information about saving your new configuration.

About the New Configuration Form

Top Cell specifies the top cellview of the design.

Library lets you type or select the name of the library that contains the top cellview.

Cell lets you type or select the cell name of the top cellview.

View lets you type or select the view name of the top cellview.

Global Bindings specifies default bindings for the entire design.

Library List lets you specify libraries for cells that do not have fixed library bindings. List the libraries in the order you want them searched. Separate entries with spaces.

View List lets you specify the views you want in your configuration in order of preference. The view list applies to every level of the configuration and determines which view is selected for every object in the design, unless overridden by a cell or instance binding.

Separate entries with spaces. You can use constants in the view list; see "Using Constants" on page 189 for more information. You can also use the asterisk character (*) as a wildcard in the view list; see "Using Wildcards in a View List" on page 193 for more information.

Stop List lets you specify the views that are to be treated as leaf nodes, that is, they are not to be expanded. Separate entries with spaces. You can use constants in the stop list; see "Using Constants" on page 189 for more information. This field can be empty.

Constraints List lets you specify a list of constraint views. The first view that is found is used. If none of the views in the list are found, the schematic editor will create a new view, of the same name as the first view in the list.

Description lets you enter a brief statement describing the configuration.

OK sets the values you entered and closes the form.

Cancel closes the New Configuration form without applying your values.

Use Template lets you select a template from a list of predefined templates. Refer to "Working with Templates" on page 106 for more information.

Help opens this manual.

Editing Configurations

This section provides an overview of how to edit a configuration. For a more detailed description on how to make changes and specify rules for configurations, see Chapter 3, "Changing Design Components.".

Note: You cannot edit sub-configurations—a sub-configuration is a configuration that is contained within a configuration. To edit a sub-configuration, you must open it separately as a configuration in the Hierarchy Editor. For more information about sub-configurations, see "About Sub-Configurations" on page 97.

To edit a configuration using the Hierarchy Editor,

1. Open the configuration.
2. Edit any of the following:
   - ❑ In the *Top Cell* section, you can change the top cellview of the configuration by editing the library name, cell name, or view name.
   - ❑ In the *Global Bindings* section, you can edit the library list, view list, stop list, and constraint list. Separate the entries in the lists with spaces.

     You can use constants in the view list and stop list. A constant is a symbolic name used to represent a set of views. For information on creating and using constants, see "Using Constants" on page 189.

     You can use the asterisk character (`*`) as a wildcard in the view list; see "Using Wildcards in a View List" on page 193 for more information.

     Note: The view list that applies to the root cellview of the design cannot be edited. In the *Cell Bindings* table, the root cellview is identified with a green pyramid in the *Info* column.
   - ❑ Edit the Cell Bindings and Instance Bindings sections.
3. Choose *View – Update* to see the results of your changes in the *Cell Bindings* or *Instance Bindings* tables.

   If you selected the *Automatic Update* option in the Options form, this step is not required because your configuration is automatically updated.
4. Click *File – Save* or press `Control-s`.

   Note: Other Cadence software tools cannot access new or changed configuration files until they have been saved.

Editing the Cell Bindings and Instance Bindings Tables

Changing Cell or Instance View Bindings

1. In the *Cell Bindings* or *Instance Bindings* table, select the cell or instance you want to change.
2. Do one of the following:
   - ❑ Click in the *View to Use* column of the cell or instance, type the new view, then press `Return`.
   - ❑ Right-click anywhere in the row of the cell or instance you want to change and select a view from the list of views in the pop-up menu.

   The new view appears in the *View to Use* column in the color used to display user bindings.
3. Choose *View – Update* to see the results of your changes.

   If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.
4. To save your changes, choose *File – Save* or press `Control-s`.

Adding Views to Cell or Instance Bindings

To add a new view,

1. In the *Cell Bindings* or *Instance Bindings* table, click in the *Inherited View List* column and move the cursor to the end of the list.
2. Type the name of the view you want to use.

   Separate entries with spaces. You can use constants in the view list; see "Using Constants" on page 189 for more information. You can also use the asterisk character (`*`) as a wildcard in the view list; see "Using Wildcards in a View List" on page 193 for more information.
3. Press `Return`.

   The modified view list appears in the *Inherited View List* column in the color used to display user bindings.
4. To view your changes, choose *View – Update*.

If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

5. To save the changes, choose *File – Save* or press `Control-s`.

Deleting Views from Cell or Instance Bindings

You can remove any views from the cell or instance bindings.

To remove a view,

1. In the *Cell Bindings* or *Instance Bindings* table, select the cell or instance you want to change.
2. In the *Inherited View List* column, double-click the view you want to remove.

   The view is highlighted.
3. Press `Delete`.
4. To view your changes, choose *View – Update*.

   If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.
5. To save the changes, choose *File – Save* or press `Control-s`.

Viewing Instances Contained by Other Instances

To see instances contained by other instances,

1. Select the *Tree View* tab to display the tree view of the Hierarchy Editor.
2. Right-click the instance you want to browse.

   A pop-up menu appears.
3. Select *Expand Instance*.

   The hierarchy below the instance is displayed.

Performing a Global Change on a Group of Instances

To perform a global view change on a group of instances, you can display all instances that share the same rules,

1. Choose *View – Options*.

The Options form appears.

2. Select the *Tree* tab.
3. In the *Expand Mode* section, select *By Instance Grouping*.
4. Click *OK*.
5. Choose *View – Tree* to display the tree view of the configuration.
6. Select the group of instances that you want to change.
7. Edit the *View to Use*, *Inherited View List*, or *Inherited LIbrary List* column for the changes you want to make to the group of instances.
8. Press `Return`.

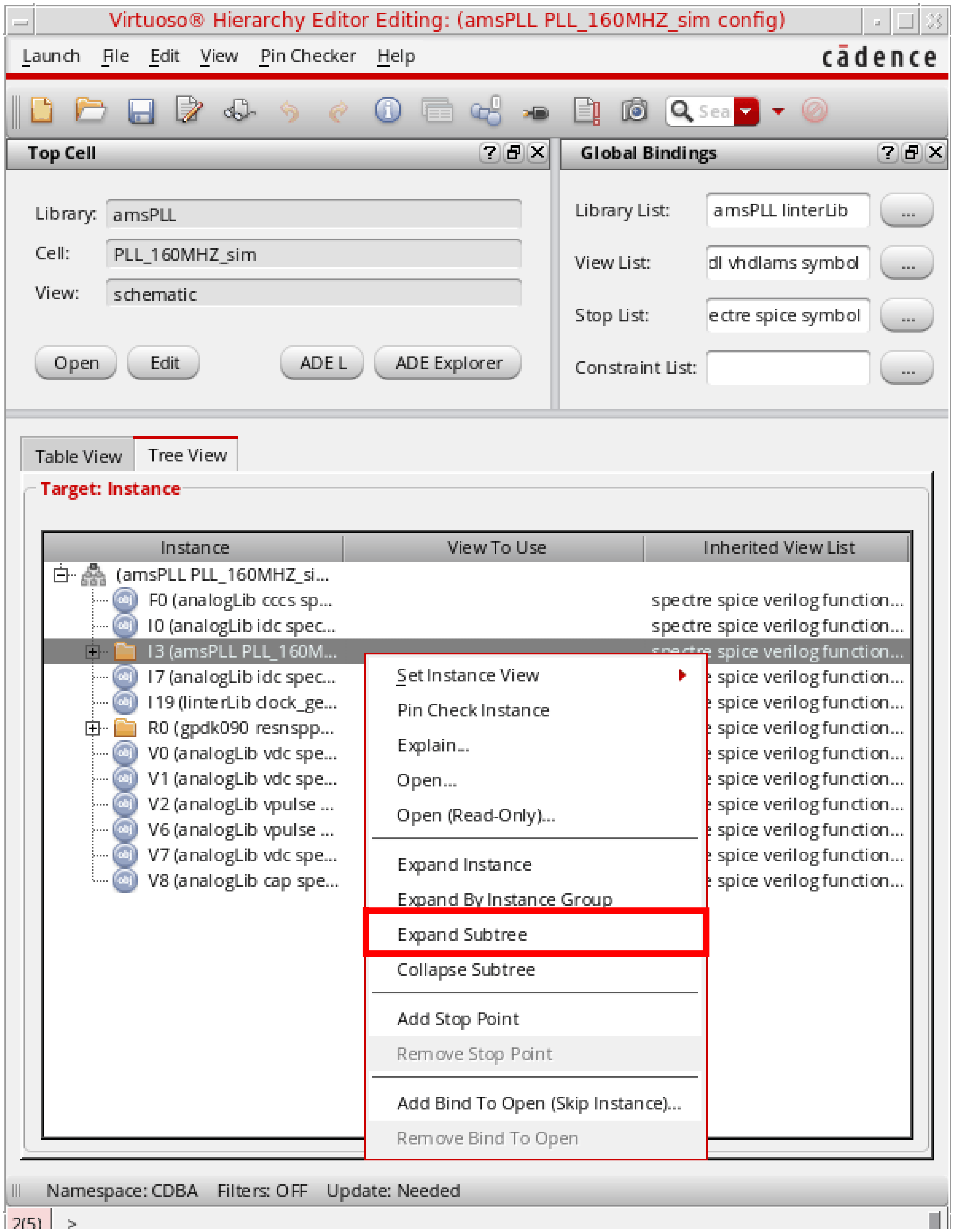
   The change you made is applied to all the instances and appears in the color used to display user bindings.
9. To see the results of your changes, choose *View – Update*.

   If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.
10. To save the changes, choose *File – Save* or press `Control-s`.

View All Instances of a Subtree

To see all the instances of a subtree:

1. Select the *Tree View* tab.
2. Right-click the instance you want to browse and select *Expand Subtree* from the context-sensitive menu.



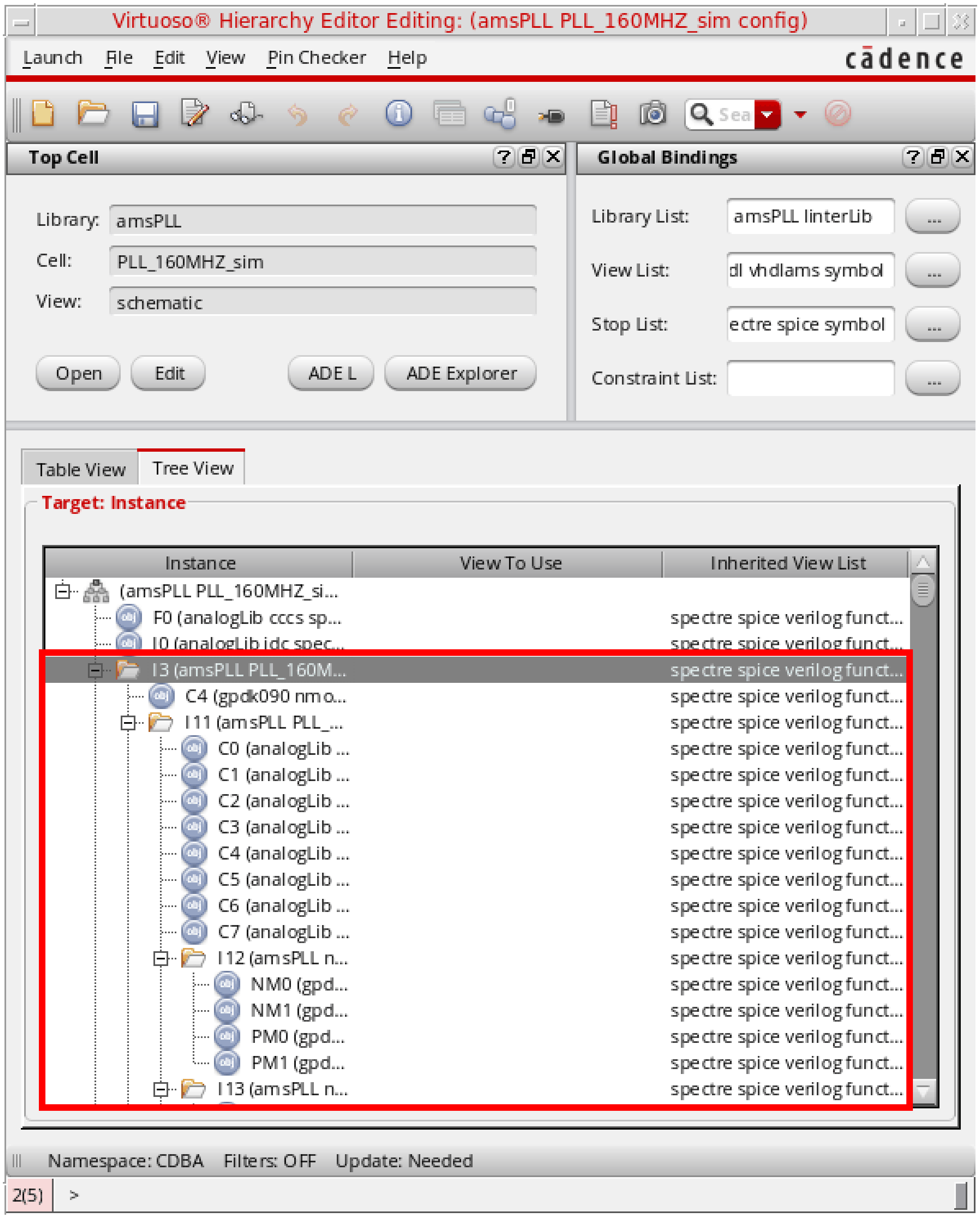
Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

Library List: amsPLL linterLib
View List: dl vhdlams symbol
Stop List: ectre spice symbol
Constraint List:

Instance	View To Use	Inherited View List
(amsPLL PLL_160MHZ_si...		
F0 (analogLib cccs sp...		spectre spice verilog function...
I0 (analogLib idc spec...		spectre spice verilog function...
I3 (amsPLL PLL_160M...		spectre spice verilog function...
I7 (analogLib idc spec...		spice verilog function...
I19 (linterLib clock_ge...		spice verilog function...
R0 (gpdk090 resnspp...		spice verilog function...
V0 (analogLib vdc spe...		spice verilog function...
V1 (analogLib vdc spe...		spice verilog function...
V2 (analogLib vpulse ...		spice verilog function...
V6 (analogLib vpulse ...		spice verilog function...
V7 (analogLib vdc spe...		spice verilog function...
V8 (analogLib cap spe...		spice verilog function...

Namespace: CDBA Filters: OFF Update: Needed

The hierarchy below the instance is displayed.



Library: amsPLL

Cell: PLL_160MHZ_sim

View: schematic

Library List: amsPLL linterLib

View List: dl vhdlams symbol

Stop List: ectre spice symbol

Constraint List:

Instance	View To Use	Inherited View List
(amsPLL PLL_160MHZ_si...		
F0 (analogLib cccs sp...		spectre spice verilog funct...
I0 (analogLib idc spec...		spectre spice verilog funct...
I3 (amsPLL PLL_160M...		spectre spice verilog funct...
C4 (gpdk090 nmo...		spectre spice verilog funct...
I11 (amsPLL PLL_...		spectre spice verilog funct...
C0 (analogLib ...		spectre spice verilog funct...
C1 (analogLib ...		spectre spice verilog funct...
C2 (analogLib ...		spectre spice verilog funct...
C3 (analogLib ...		spectre spice verilog funct...
C4 (analogLib ...		spectre spice verilog funct...
C5 (analogLib ...		spectre spice verilog funct...
C6 (analogLib ...		spectre spice verilog funct...
C7 (analogLib ...		spectre spice verilog funct...
I12 (amsPLL n...		spectre spice verilog funct...
NM0 (gpd...		spectre spice verilog funct...
NM1 (gpd...		spectre spice verilog funct...
PM0 (gpd...		spectre spice verilog funct...
PM1 (gpd...		spectre spice verilog funct...
I13 (amsPLL n...		spectre spice verilog funct...

Namespace: CDBA Filters: OFF Update: Needed

2(5) >

Now, to close the hierarchy, right-click the instance and select *Collapse Subtree* from the context-sensitive menu.

About Sub-Configurations

A sub-configuration is a configuration that is used within another configuration. A sub-configuration is read-only within the configuration.

Cells contained in a sub-configuration are also displayed in the *Cell Bindings* table and are identified with an asterisk symbol in the *Information* column. The root cellview of the sub-configuration is identified with a brown pyramid in the *Information* column of the Cell Bindings table as well as in the Tree view.

You create a sub-configuration the way you create any other configuration (see "Creating a New Configuration" on page 86 for more information).

You can then use the sub-configuration in your configuration in the same way that you use any other cellview.

Note: After you have set a cell or instance to a sub-configuration, an update may need to be performed to see the latest contents of the sub-configuration.

To change the representation of a cell in your configuration to a sub-configuration

➤ Set the view of the cell or its instance to the configuration view by doing one of the following:

  ❑ Specify the configuration view in the *View to Use* field of the cell or instance.

  ❑ Right-click the cell or instance and use the *Set Cell View* or *Set Instance View* command on the pop-up to select the configuration view.

  The cell or instance is then identified as a sub-configuration within your configuration.

A sub-configuration is read-only—you cannot edit it from the configuration it is contained in.

To edit a sub-configuration

➤ Open the sub-configuration in the Hierarchy Editor with the *File – Open* command and then edit it.

Verifying Configuration Data

You can check the configuration data of your design in the following ways:

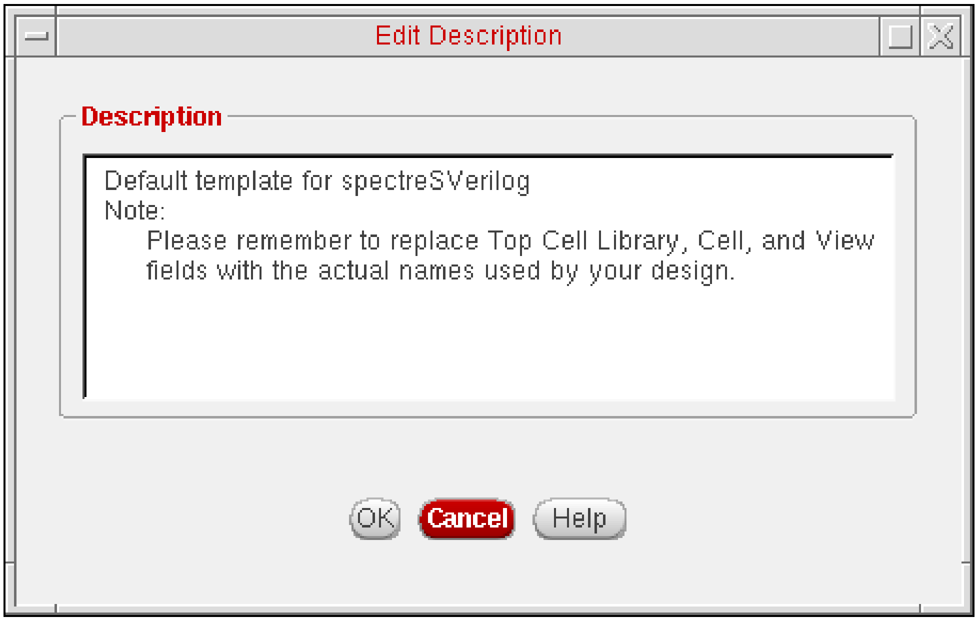
- Viewing the Description File
- Seeing what binding rules were used to select any view in the hierarchy. You can do this by
  - Verifying Cell Binding Rules
  - Verifying Instance Binding Rules
  - Verifying Occurrence Binding Rules
- Looking at the structure of the hierarchy using the table or the tree view.

Viewing the Description File

To view the description file,

➤ Choose *Edit – Description*.

The Edit Description form appears.



Default template for spectreSVerilog
Note:
Please remember to replace Top Cell Library, Cell, and View fields with the actual names used by your design.

Description displays the current description of the configuration, which you can edit.

OK applies the changes you made and closes the form.

Cancel cancels the changes you made and closes the form.

Help opens this manual.

To edit the description of your configuration,

1. In the Edit Description form, edit the description in the *Description* field.
2. Click *OK*.

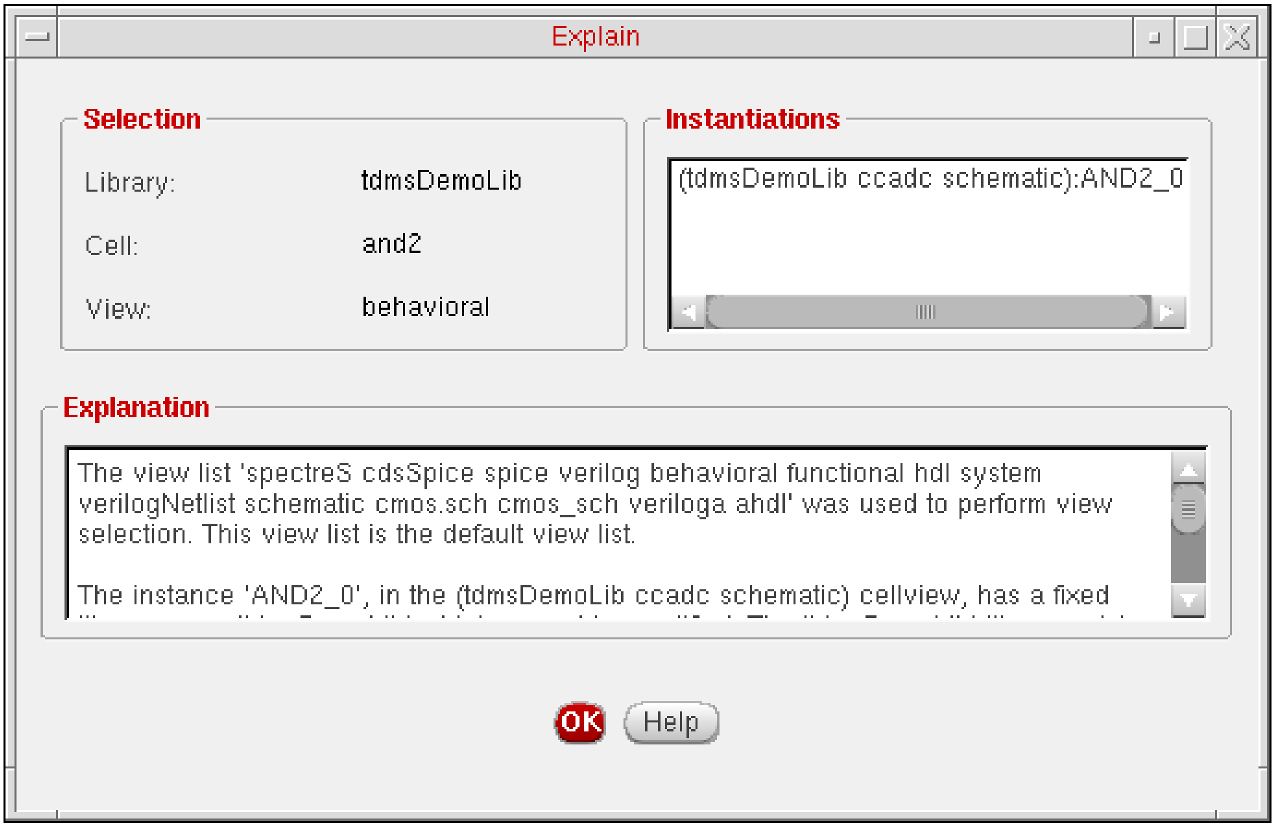
   The Hierarchy Editor saves the edited description.

Verifying Cell Binding Rules

To see which binding rules were used for a cell, do one of the following:

- In the *Cell Bindings* table, right-click a cellview and, from the pop-up menu, select *Explain*.
- In the *Cell Bindings* table, select a cellview and, on the toolbar, click the *Explain* icon.

The Explain form is displayed.



Library: tdmsDemoLib
Cell: and2
View: behavioral

(tdmsDemoLib ccadc schematic):AND2_0

The view list 'spectreS cdsSpice spice verilog behavioral functional hdl system verilogNetlist schematic cmos.sch cmos_sch veriloga ahdl' was used to perform view selection. This view list is the default view list.

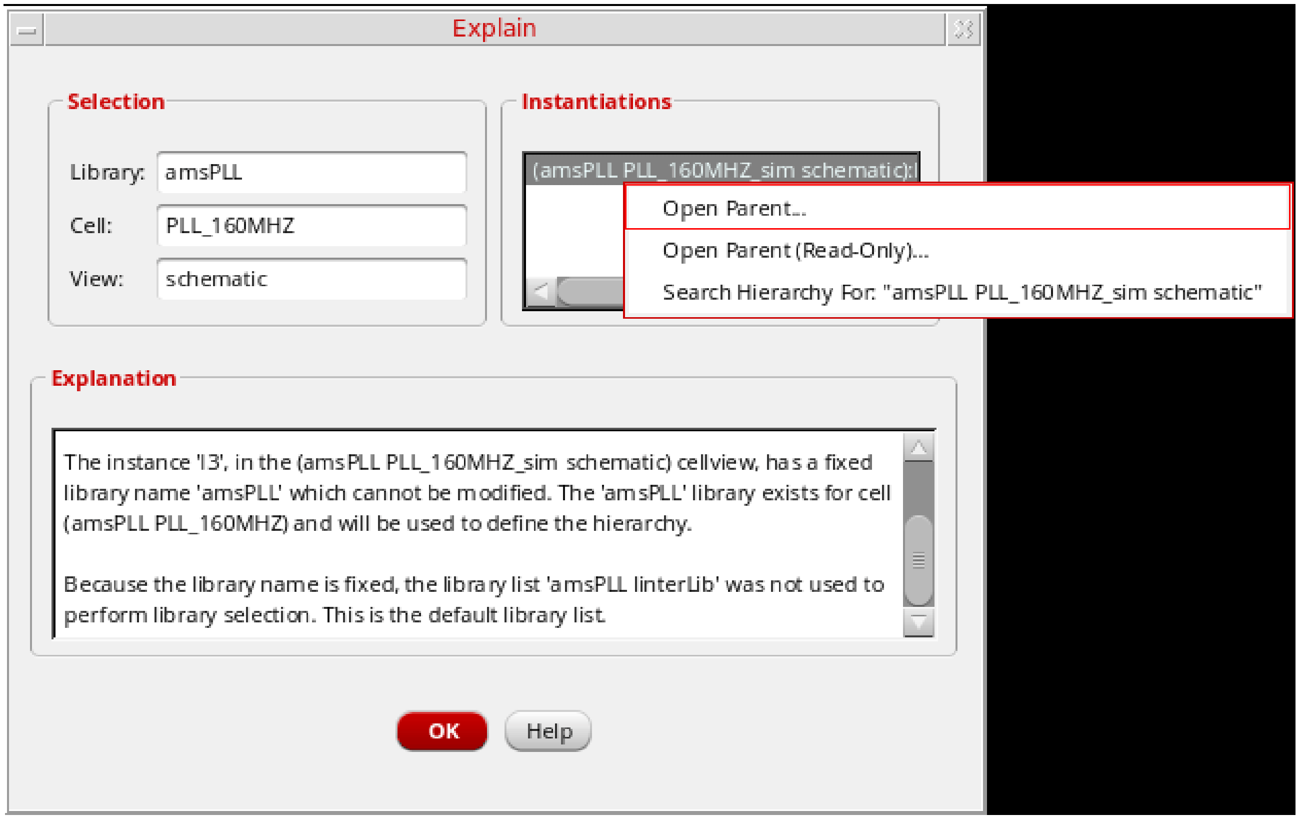
The instance 'AND2_0', in the (tdmsDemoLib ccadc schematic) cellview, has a fixed

About the Explain Form for a Selected Cell

Selection displays the master cellview for the selected cell.

> **Library** specifies the master library of the selected cell.
>
> **Cell** specifies the master cell of the selected cell.
>
> **View** specifies the master view of the selected cell.
>
> **Note:** If the *Library*, *Cell*, or *View* fields display `**UNBOUND**`, the cell has a bind-to-open attribute.

Instantiations lists all the instantiations of the cell. You can also open the parent cellview of the selected instance by clicking the Right Mouse Button and selecting *Open Parent* from the context-sensitive menu.


Explain
Selection
Instantiations
Open Parent...
Open Parent (Read-Only)...
Search Hierarchy For: "amsPLL PLL_160MHZ_sim schematic"
Explanation
OK
Help

Library: amsPLL

Cell: PLL_160MHZ

View: schematic

(amsPLL PLL_160MHZ_sim schematic):I

The instance 'I3', in the (amsPLL PLL_160MHZ_sim schematic) cellview, has a fixed library name 'amsPLL' which cannot be modified. The 'amsPLL' library exists for cell (amsPLL PLL_160MHZ) and will be used to define the hierarchy.

Because the library name is fixed, the library list 'amsPLL linterLib' was not used to perform library selection. This is the default library list.

This option allows opening the parent cellview of the selected instance (out of context) or the selected occurrence in the context of the current configuration.

Explanation describes the bindings for the selected cell.

OK closes the form.

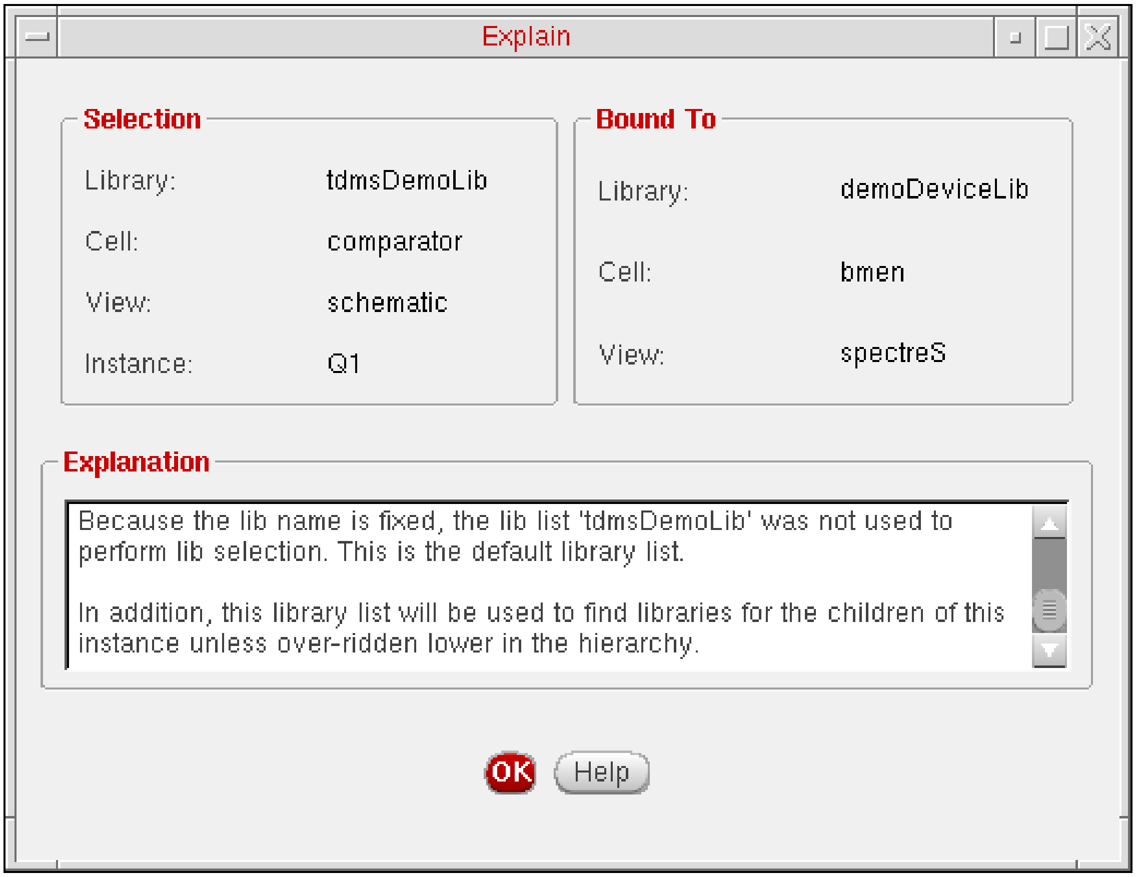
Help opens this manual.

Verifying Instance Binding Rules

To see which binding rules were used for an instance, do one of the following:

- In the *Instance Bindings* table, right-click the instance whose rules you want to check and, from the pop-up menu, select *Explain*.
- In the *Instance Bindings* table, select the instance and, on the toolbar, click the *Explain* icon.

The Explain form appears.



Selection

Library:	tdmsDemoLib
Cell:	comparator
View:	schematic
Instance:	Q1

Bound To

Library:	demoDeviceLib
Cell:	bmen
View:	spectreS

Explanation

Because the lib name is fixed, the lib list 'tdmsDemoLib' was not used to perform lib selection. This is the default library list.

In addition, this library list will be used to find libraries for the children of this instance unless over-ridden lower in the hierarchy.

About the Explain Form for a Selected Instance

Selection displays the cellview that contains the selected instance

Library specifies the library of the cellview that contains the selected instance.

Cell specifies the cell of the cellview that contains the selected instance.

View specifies the view name of the cellview that contains the selected instance.

Instance specifies the name of the instance in the cellview that contains it.

Bound To displays the results of the rules that are used by the selected instance.

Library specifies the master library of the selected instance.

Cell specifies the master cell of the selected instance.

View specifies the master view of the selected instance.

Note: If the *Library*, *Cell*, or *View* fields display `**UNBOUND**`, the instance has a bind-to-open attribute.

Explanation describes the bindings for the selected instance.

OK closes the form.

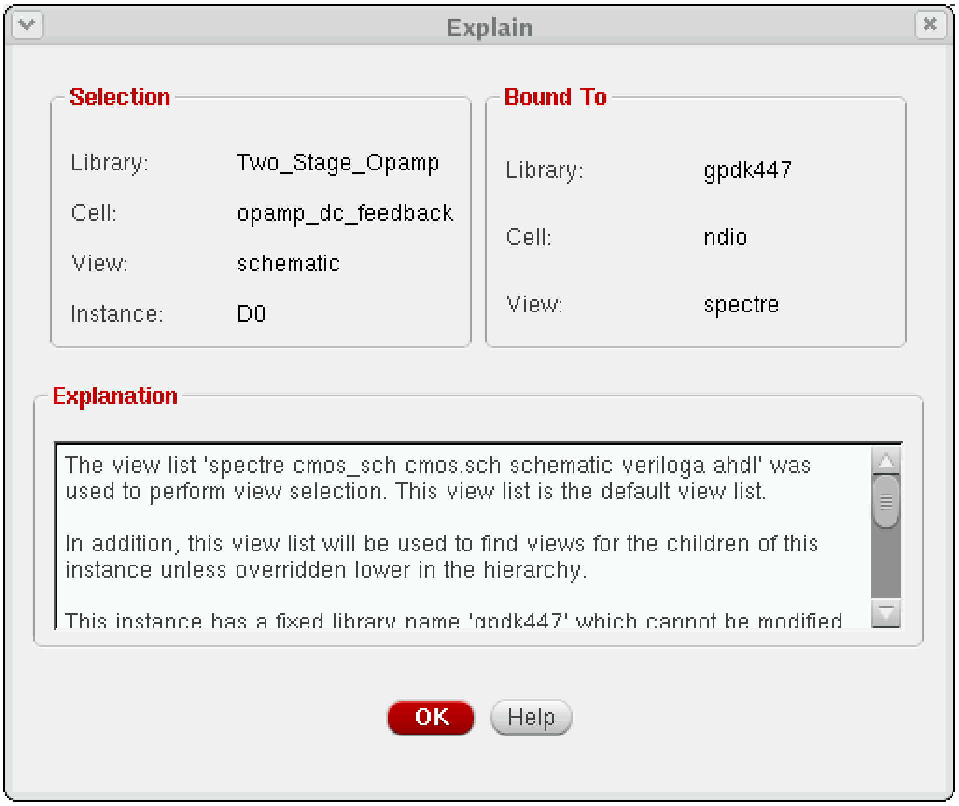
Help opens this manual.

Verifying Occurrence Binding Rules

To see which binding rules were used for an occurrence,

1. Click the *Tree View* tab to display the tree view of the configuration.
2. Right-click the occurrence whose rules you want to check.
3. From the pop-up menu, select *Explain*.

The Explain form appears.



Selection

Library: Two_Stage_Opamp
Cell: opamp_dc_feedback
View: schematic
Instance: D0

Bound To

Library: gpdk447
Cell: ndio
View: spectre

Explanation

The view list 'spectre cmos_sch cmos.sch schematic veriloga ahdl' was used to perform view selection. This view list is the default view list.

In addition, this view list will be used to find views for the children of this instance unless overridden lower in the hierarchy.

This instance has a fixed library name 'gpdk447' which cannot be modified

About the Explain Form for a Selected Occurrence

Selection displays the cellview that contains the selected occurrence

Library specifies the library of the cellview that contains the selected occurrence.

Cell specifies the cell that contains the selected occurrence.

View specifies the view name of the cellview that contains the selected occurrence.

Instance specifies the name of the instance in the cellview that contains the occurrence.

Bound To displays the library, cell, and view to which the occurrence is bound.

Library specifies the library of the selected occurrence.

Cell specifies the cell of the selected occurrence.

View specifies the view of the selected occurrence.

> **Note:** If the *Library*, *Cell*, or *View* fields display `**UNBOUND**`, the occurrence has a bind-to-open attribute.

Explanation describes the bindings for the selected occurrence.

OK closes the form.

Help opens this manual.

Working with Templates

Templates let you build configurations using pre-defined view lists, library lists, and stop lists. The Hierarchy Editor includes templates for some simulators that are compatible with Cadence software. You can also create your own templates.

Templates are located using CSF (Cadence Search File mechanism). The CSF search will include those templates in the installation hierarchy, and also any user-created templates that are found under `hierEditor/templates`, in directories that have been defined in `setup.loc`, for example `/hierEditor/templates/mytemplate`.

If any templates are found in one of these locations, they will be displayed in the *Name* drop-down, without the need to specify a path.

Note: If a template is not found in one of the locations mentioned, you can enter a path in the *From File* field.

Using Templates

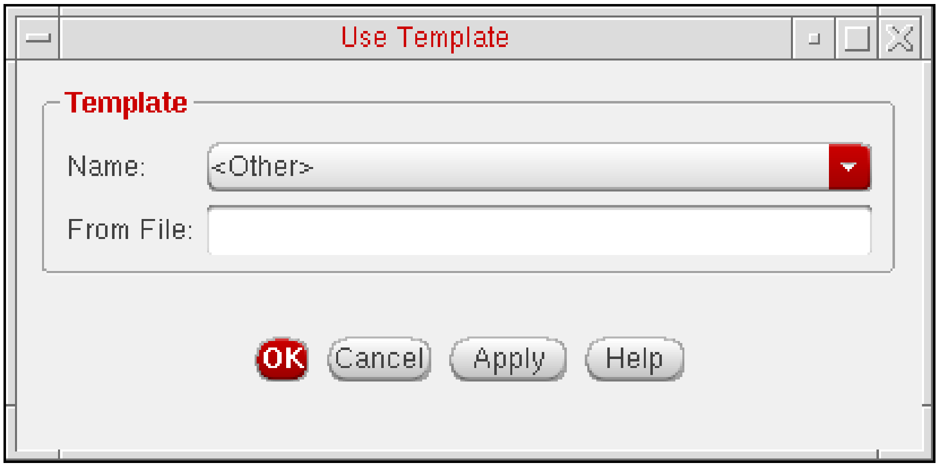
To use a template,

1. Choose *File – New Config* or press `Control-n`.

   The New Configuration form appears.

2. In the New Configuration form, click *Use Template*.

   The Use Template form appears.



3. Do one of the following:

- ❑ If you want to use a pre-defined template, or a template that you have created and placed in the `your_install_dir/share/cdssetup/hierEditor/templates` directory, select the template from the *Name* listbox.

  Note: Template options available, dependent upon what tools are installed, are: *AMS*, *auCdl*, *auLvs*, *hSpiceD*, *spectre*, *spectreVerilog*, *systemVerilog*, *verilog*, and *vhdlInteg*.

- ❑ If you want to use a template that is not in the `your_install_dir/share/cdssetup/hierEditor/templates` directory,

  - ❍ Select *<Other>* from the *Name* listbox.
  - ❍ In the *From File* field, type the path to the template.

4. Click *OK*.

The Use Template form closes. The New Configuration form is filled in with the library list, view list, stop list, and constraint list data from the template.

5. In the *Top Cell* section, specify a top cellview for the new configuration.

6. Click *OK*.

The Hierarchy Editor displays the new configuration.

About the Use Template Form

Template

Name lets you select a template from a list of pre-defined templates for simulators that are compatible with Cadence software. The listbox also lists any other templates that you placed in the `your_install_dir/share/cdssetup/hierEditor/templates/` directory. If you want to use a template that is not in this directory, select *<Other>* and specify the path in the *From File* field.

From File lets you type the path to the template you want to use.

OK applies your settings and closes the form. The information from the template you selected is displayed in the New Configuration form.

Cancel closes the form without applying your settings.

Apply sets your selections and leaves the form open.

Help opens this manual.

Creating Templates

You can create your own template using your own view lists, library lists, stop lists, and constraints lists for your simulator or other design-specific requirements.

To create your own template,

1. In the Hierarchy Editor, choose *File – New Config* or press `Control-n`.

   Note: If you have made changes to the current configuration, the Hierarchy Editor prompts you to save your changes.

   The New Configuration form appears.

2. In the *Top Cell* section, type the library, cell, and view name for the top cellview. These names are placeholders—you will replace them when you use the template.

3. In the *Global Bindings* section, specify the library list, view list, stop list, and constraints list.

   Separate entries with spaces.

   You can use constants in the view list and stop list; see “Using Constants” on page 189 for more information. You can also use the asterisk character (`*`) as a wildcard in the view list; see “Using Wildcards in a View List” on page 193 for more information.

4. (Optional) In the *Description* field, specify a brief description of the new template.

5. Click *OK*.

   The New Configuration form closes.

   The Hierarchy Editor displays the new configuration.

6. Choose *File – Save As*.

   The Save As form appears.

7. In the Save As form, select or type the name of the library, cell, and view in which you want to save the new template.

   Note: The names do not have to match the names of the cell and view of the top-level cell.

8. Click *OK*.

   The Save As form closes.

9. Open a terminal window.

10. Create a directory in which you want to place your templates.
11. Change directories to the `/library/cell/view` directory in which you saved the template.
12. Copy the configuration file in the directory (the `expand.cfg` file) to the directory you created. For example:

    ```
    cp expand.cfg yourTemplatesDirectoryPath
    ```

When you use the new template, you will need to specify the path to the template (in the *From File* field of the Use Template form).

You can also copy your template to the location in which pre-defined Hierarchy Editor templates are stored.

Copying Your New Template to the Pre-defined Templates Location

You can copy the templates you create to the directory that contains pre-defined Hierarchy Editor templates and access them from the *Use Template* button on the *New Configuration* form.

To copy your template,

1. Change directories to the `/library/cell/view` directory in which you saved the template.
2. Copy the configuration file to the templates directory with the following command:

   ```
   cp expand.cfg your_install_dir/share/cdssetup/hierEditor/templates/
   yourtemplatename
   ```

3. Restart the Hierarchy Editor.

   Your new template is now available from the *Use Template* button on the *New Configuration* form.

Saving Configurations

When you save a configuration in the Hierarchy Editor, it is saved in the `expand.cfg` file in a configuration view. The configuration view directory also contains an `expand.cfg%` file, which is the previously-saved version of the `expand.cfg` file.

You can also save configurations in VHDL and Verilog®.

Note: For VHDL, that is in VHDL syntax, while for Verilog, it is in a file that contains paths to Verilog files (so that xmvlog can process them with the `-file (-f)` option).

See the following topics for more information:

- "Saving a Configuration as VHDL" on page 114
- "Saving a Configuration as Verilog" on page 118

If you are a Virtuoso® Schematic Editor user, see "Using the Virtuoso Schematic Editor with the Hierarchy Editor" on page 132.

Note: Other applications cannot access changed configuration files until they have been saved in the Hierarchy Editor.

Saving a Configuration

To save a configuration,

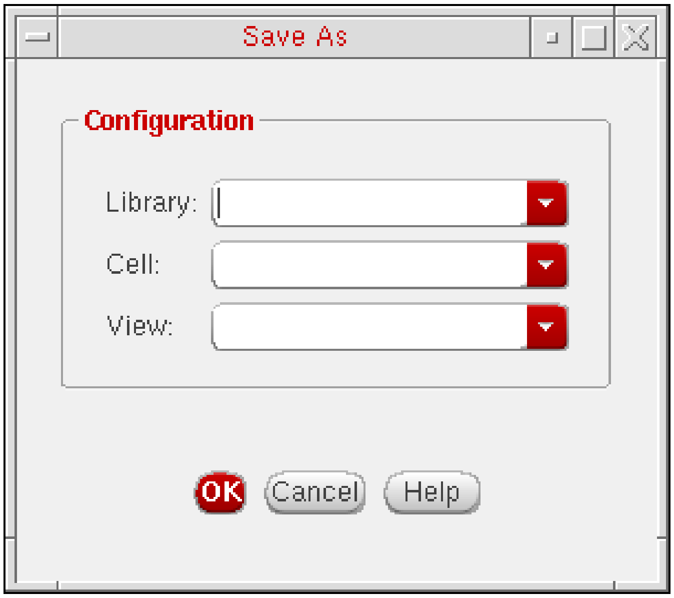
➤ Choose *File – Save* or press `Control-s`.

The Hierarchy Editor saves the configuration.

To save a configuration using the *File – Save As* command,

1. Choose *File – Save As*.

The Save As form appears.



(For details about the Save As form, see “About the Save As Form” on page 113.)

2. In the *Library* field, type or select the library in which you want to save the configuration.
3. In the *Cell* field, type or select the cell in which you want to save the configuration.
4. In the *View* field, type the name of the new configuration view.

   Note: The name you give to the cell and view of the configuration do not have to match the name of the cell and view of the top-level cellview.

5. Click *OK*.

About the Save As Form

Configuration

Library lets you specify the name of the library in which you want to save a copy of the configuration.

Cell lets you specify the name of the cell in which you want to save the configuration.

View lets you specify the name of the new configuration view.

OK saves the configuration to the new name.

Cancel cancels your selections and closes the form.

Help opens this manual.

Saving a Configuration as VHDL

The Hierarchy Editor provides the *File – Save As VHDL* command to save a configuration in VHDL syntax so that it can be read by VHDL tools. The VHDL configuration is always saved in a view called `configuration`.

The Hierarchy Editor adds those cellviews that have a VHDL source file to the VHDL configuration. It currently considers `vhdl.vhd` and `vhdl.vams` files as VHDL source files.

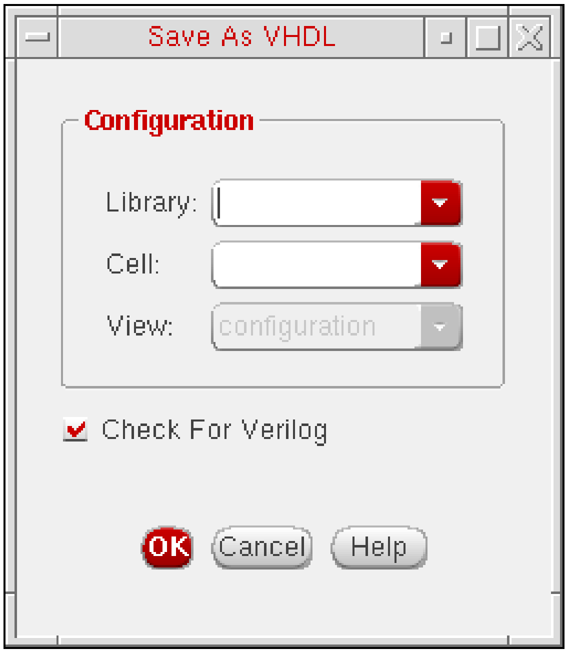
The Hierarchy Editor also adds those cellviews that do not have VHDL source files to the configuration but comments them out and generates a warning for them. However, if these cellviews have Verilog source files, you can choose to add them to the VHDL configuration by selecting the *Check for Verilog* option in the Save As VHDL form. If you select this option, the Hierarchy Editor adds the cellviews that do not have VHDL source files but do have Verilog source files to the VHDL configuration without commenting them out and does not generate a warning for them.

Note: If Verilog views are included in the VHDL configuration based on the above, the inner `for/end` statements for these views are not written out.

To save a configuration for VHDL applications,

1. Choose *File – Save As VHDL*.

   The Save As VHDL form appears.



Save As VHDL

Configuration

Library:

Cell:

View: configuration

☑ Check For Verilog

2. In the *Library* and *Cell* fields, specify the name of the library and cell in which you want to save the VHDL configuration.

   The *View* field is automatically set to `configuration`.

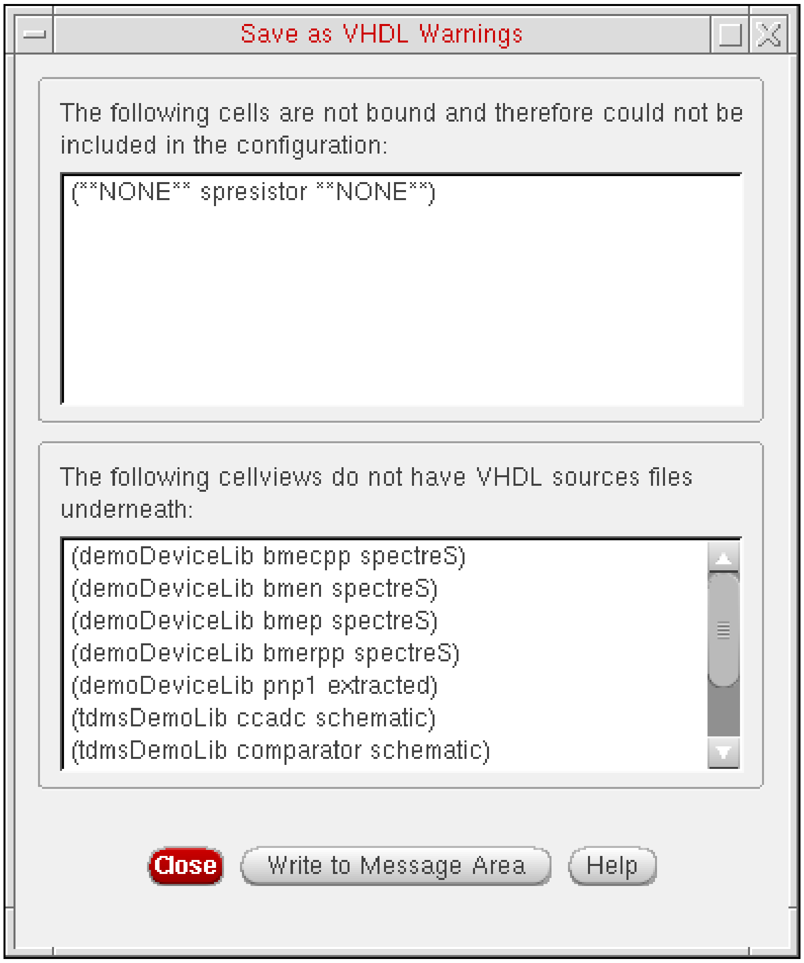
3. Select the *Check for Verilog* option if you want those cellviews that do not have VHDL source files but do have Verilog source files to be added to the VHDL configuration.

   If you do not select this option, the Hierarchy Editor adds all the cellviews that do not have VHDL source files to the VHDL configuration but comments them out and generates a warning for them.

4. Click *OK*.

   Note: Saving a VHDL configuration does not save it in a format that can be read and edited by the Hierarchy Editor. You must also save the configuration with the *File – Save* or *File – Save As* command in order for it to be read by the Hierarchy Editor.

The VHDL configuration is saved. If the Hierarchy Editor generated any warnings while creating the VHDL configuration, it displays a dialog box asking you whether you want to see the warnings. If you click *Yes*, the Hierarchy Editor displays the following dialog box:



The following cells are not bound and therefore could not be included in the configuration:

```
("**NONE** spresistor **NONE**)
```

The following cellviews do not have VHDL sources files underneath:

```
(demoDeviceLib bmecpp spectreS)
(demoDeviceLib bmen spectreS)
(demoDeviceLib bmep spectreS)
(demoDeviceLib bmerpp spectreS)
(demoDeviceLib pnp1 extracted)
(tdmsDemoLib ccadc schematic)
(tdmsDemoLib comparator schematic)
```

The Save as VHDL Warnings dialog box contains all the warnings that were generated while the VHDL configuration was created. For example, it displays a list of cells that are unbound and therefore not included in the configuration or a list of cellviews that do not have VHDL source files.

Note: If you selected the *Check for Verilog* option when you saved the VHDL configuration, the warnings dialog box does not list those cellviews that do not have VHDL source files but do have Verilog source files.

To save the warnings to a log file,

➤ Click *Write to Message Area*.

The warnings are displayed in the Messages area and also saved to the log file.

Important

If you access the standalone version of the Hierarchy Editor, it will contain a designated message area at the bottom of the window. This is not however required in the Virtuoso-integrated version, where the Hierarchy Editor utilizes the CIW message area.

About the Save As VHDL Form

Configuration

Library lets you type or select the library in which you want to save the VHDL configuration.

Cell lets you type or select the cell in which you want to save the VHDL configuration.

View is the configuration view. For VHDL configurations, the view name is always `configuration`.

Check for Verilog lets you add cellviews that do not have VHDL source files but do have Verilog source files to the VHDL configuration. Select this option if you want to add these cellviews to the VHDL configuration; deselect it if you do not want to add these cellviews to the VHDL configuration.

Note: If the *Check for Verilog* option is deselected, the Hierarchy Editor adds all cellviews that do not have VHDL source files to the configuration but comments them out and generates a warning for them in the Save as VHDL Warnings dialog box.

OK saves the configuration.

Cancel discards your selections and closes the form.

Help opens this manual.

Saving a Configuration as Verilog

The Hierarchy Editor provides the *File – Save As Verilog* command to save a configuration to a file that can be used with the `-file` argument of Verilog applications such as xmvlog. The Verilog file can have any name, though it is typically called `verilog.f`.

The Hierarchy Editor adds a list of Verilog source files to the Verilog file.

For any cellview, if the cellview is a Verilog view (as defined by the `master.tag` file), then the Verilog source file specified in the `master.tag` is used. (If the library has a `TMP` directory associated with it, then the Hierarchy Editor looks for the source file in the temporary directory first, and then, if it is not found, looks for it in the master location.)

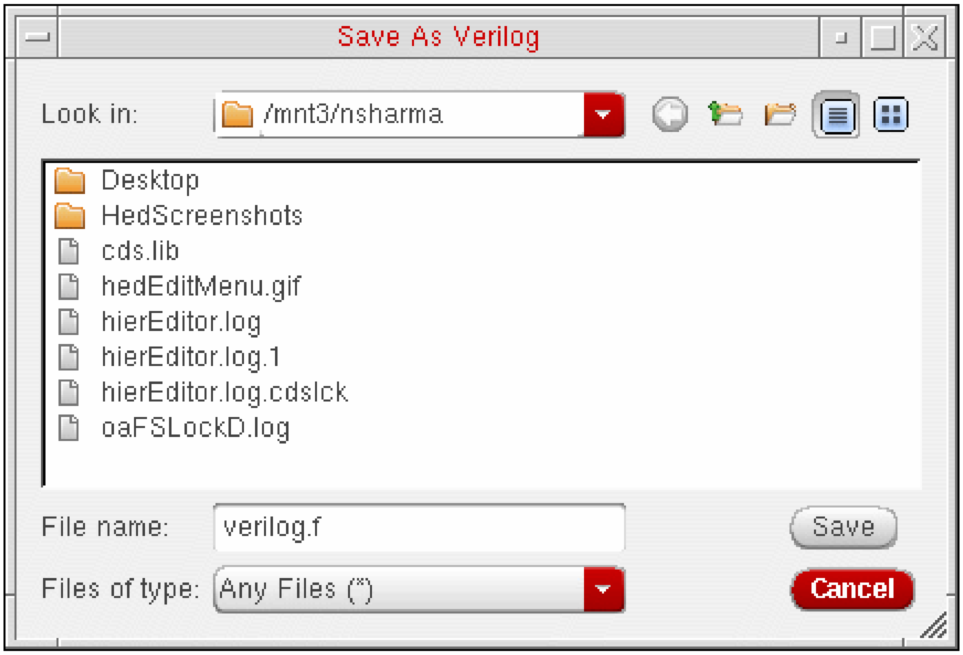
If the cellview is not a Verilog view (for example, if it is a schematic), then the Hierarchy Editor determines which Verilog source file to use. If the library has a `TMP` directory associated with it, the Hierarchy Editor first looks for source files in the temporary directory and then, if it does not find any, looks for them in the master location. If there are multiple source files, then the Hierarchy Editor selects the file according to the following order of precedence (highest to lowest):

```
verilog.vams
verilog.va
veriloga.va
verilog.v
```

To save a configuration for Verilog applications,

1. Choose *File – Save As Verilog*.

   The Save As Verilog form appears.



Save As Verilog

Look in: /mnt3/nsharma

- Desktop
- HedScreenshots
- cds.lib
- hedEditMenu.gif
- hierEditor.log
- hierEditor.log.1
- hierEditor.log.cdslck
- oaFSLockD.log

File name: verilog.f

Files of type: Any Files (*)

2. Specify the file in which you want to save the configuration by doing the following:

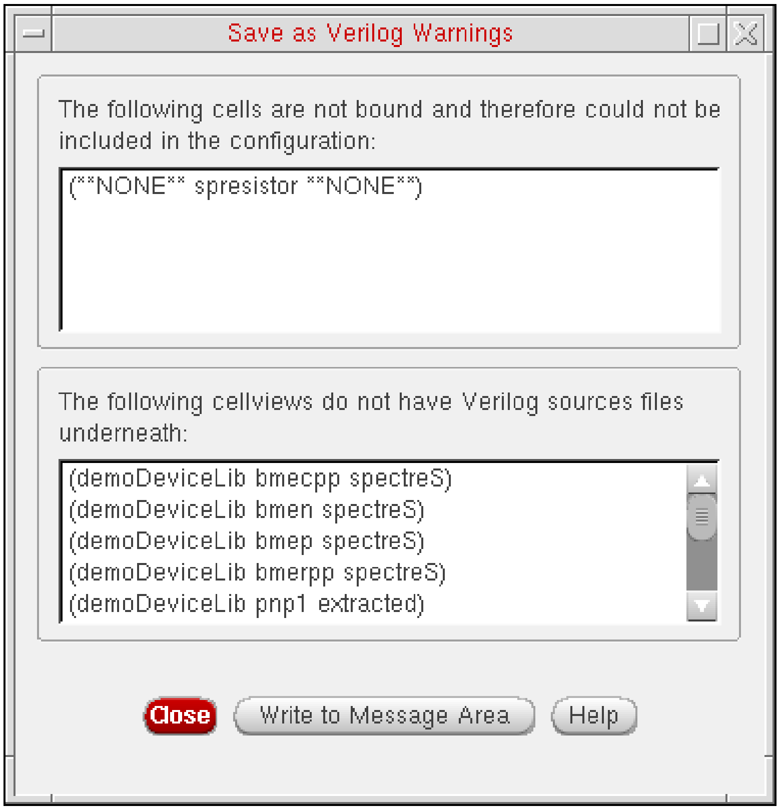
   a. In the *Look in* field, select the directory in which you want to save the file.

   b. In the *File* field, type the name of the file.

      The default value of this field is `verilog.f`.

3. Click *Save*.

The Verilog file is saved. If the Hierarchy Editor generated any warnings while creating the Verilog file, it displays a dialog box asking you whether you want to see the warnings. If you click *Yes*, the Hierarchy Editor displays the following dialog box:



The following cells are not bound and therefore could not be included in the configuration:

```
(**NONE** spresistor **NONE**)
```

The following cellviews do not have Verilog sources files underneath:

```
(demoDeviceLib bmecpp spectreS)
(demoDeviceLib bmen spectreS)
(demoDeviceLib bmep spectreS)
(demoDeviceLib bmerpp spectreS)
(demoDeviceLib pnp1 extracted)
```

The Save as Verilog Warnings dialog box contains all the warnings that were generated while the Verilog file was created. For example, it displays a list of cells that are unbound and therefore not included in the Verilog file or a list of cells that do not have Verilog source files.

To save the warnings to a log file,

➤ Click *Write to Message Area*.

The warnings are displayed in the Messages area and also saved in the log file.

Important

If you access the standalone version of the Hierarchy Editor, it will contain a designated message area at the bottom of the window. This is not however required in the Virtuoso-integrated version, where the Hierarchy Editor utilizes the CIW message area.

About the Save As Verilog Form

Look in lets you select the directory in which you want to save the configuration.

File lets you specify a name for the Verilog file. The default name for the Verilog file is `verilog.f`.

Files of type lets you narrow the files that are displayed by choosing to display only certain types of files.

Save saves the Verilog file and closes the form.

Cancel closes the form without applying your selections.

Help opens this manual.

Comparing Configurations

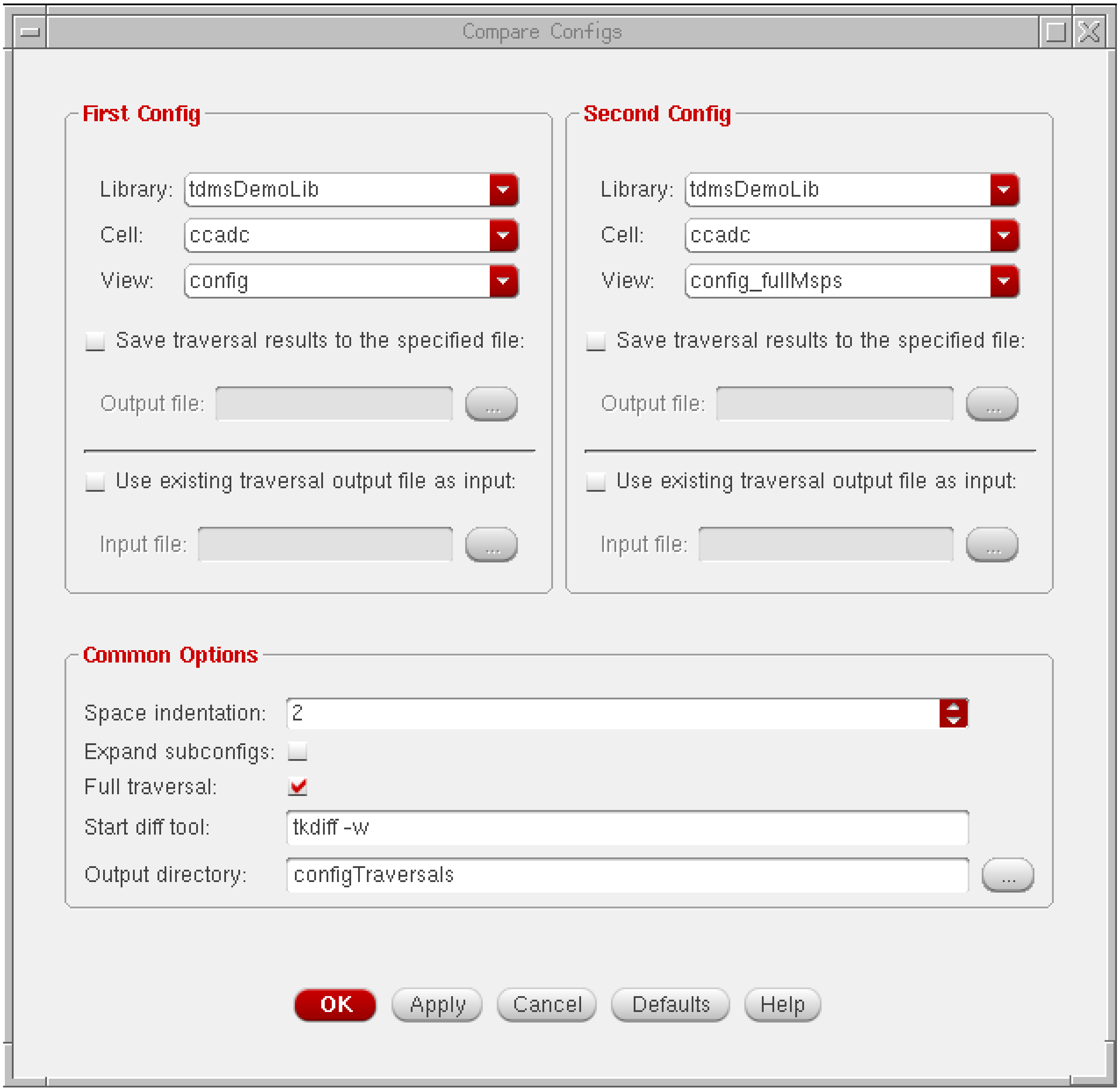
You can compare two configurations and highlight their differences by opening the resulting design hierarchy side-by-side in a tree view.

To perform the comparison, you specify two configurations that are to be compared and traverse through their hierarchies. The traversal results are saved in the traversal output files, which are then compared to highlight differences between them.

Perform the following step to compare configurations:

➡ Choose *File – Compare Configs*.

The *Compare Config* form appears as shown in the figure below:



First Config

Library: tdmsDemoLib

Cell: ccadc

View: config

□ Save traversal results to the specified file:

Output file:

□ Use existing traversal output file as input:

Input file:

Second Config

Library: tdmsDemoLib

Cell: ccadc

View: config_fullMsps

□ Save traversal results to the specified file:

Output file:

□ Use existing traversal output file as input:

Input file:

Common Options

Space indentation: 2

Expand subconfigs: □

Full traversal: ☑

Start diff tool: tkdiff -w

Output directory: configTraversals

This form includes the following sections:

- In the *First Config* section, specify the following fields:
  - ❑ *Library*—Specify the library for the first configuration.
  - ❑ *Cell*—Specify the cell.
  - ❑ *View*—Specify the view.

- ❑ *Save traversal results to the specified file:*—Select this checkbox if you want to specify an output file in which traversal results are saved. If you do not select this option, the traversal files are saved in the default traversal output directory named `configTraversals`. The traversal files are saved with the following naming convention:

  `<lib>.<cell>.<view>`

  For example, `tdmsDemoLib.ccadc.config`

  where, `tdmsDemoLib` is a library, `ccadc` is a cell, and `config` is a view.

- ❑ *Output file:* Browse and select the output file in which you want to save the results. If the output file already exists, a warning message appears to confirm overwriting the file.

- ❑ *Use existing traversal file as an input:* Select this checkbox if you do not want to perform the traversal for the configuration and want to provide an existing traversal file as an input in the current comparison. When you select this option, all the above described options are disabled.

- ❑ *Input file*—Browse and select the existing traversal file to be used for the comparison. You can directly provide the path to the output file in this field that is to be used as an input file in the comparison.

Note: If you have already opened a configuration, the configuration information is populated automatically in the *First Config* section.

- ■ In the *Second Config* section, provide the similar information as that of the first configuration.

- ■ In the *Common Options* section, specify the following fields:

  - ❑ *Space Indentation*—Specifies the number of spaces to be used for indentation. This help improve the readability of the output files.
    Default value: `2`
    Value range: `1` to `8`

  - ❑ *Expand Subconfigs*—Select this check box if you also want HED to traverse through the subconfigurations found in the configuration. This field is not selected by default.

  - ❑ *Full traversal*—Select this check box if you want to perform a full traversal, which means when a cell is instantiated several times, each instance of this cell is fully traversed and a tree view for each instance of this cell is written out to an output file. If you do not select this option, only the first visited instance of this cell is fully traversed. This field is selected by default.

- ❑ *Start diff tool*—Specify a tool that you want to use to compare the traversal results. By default, the `tkdiff` tool is used for comparison.

  Note: Ensure that `tkdiff` is available in the `$PATH` environment variable.

Important

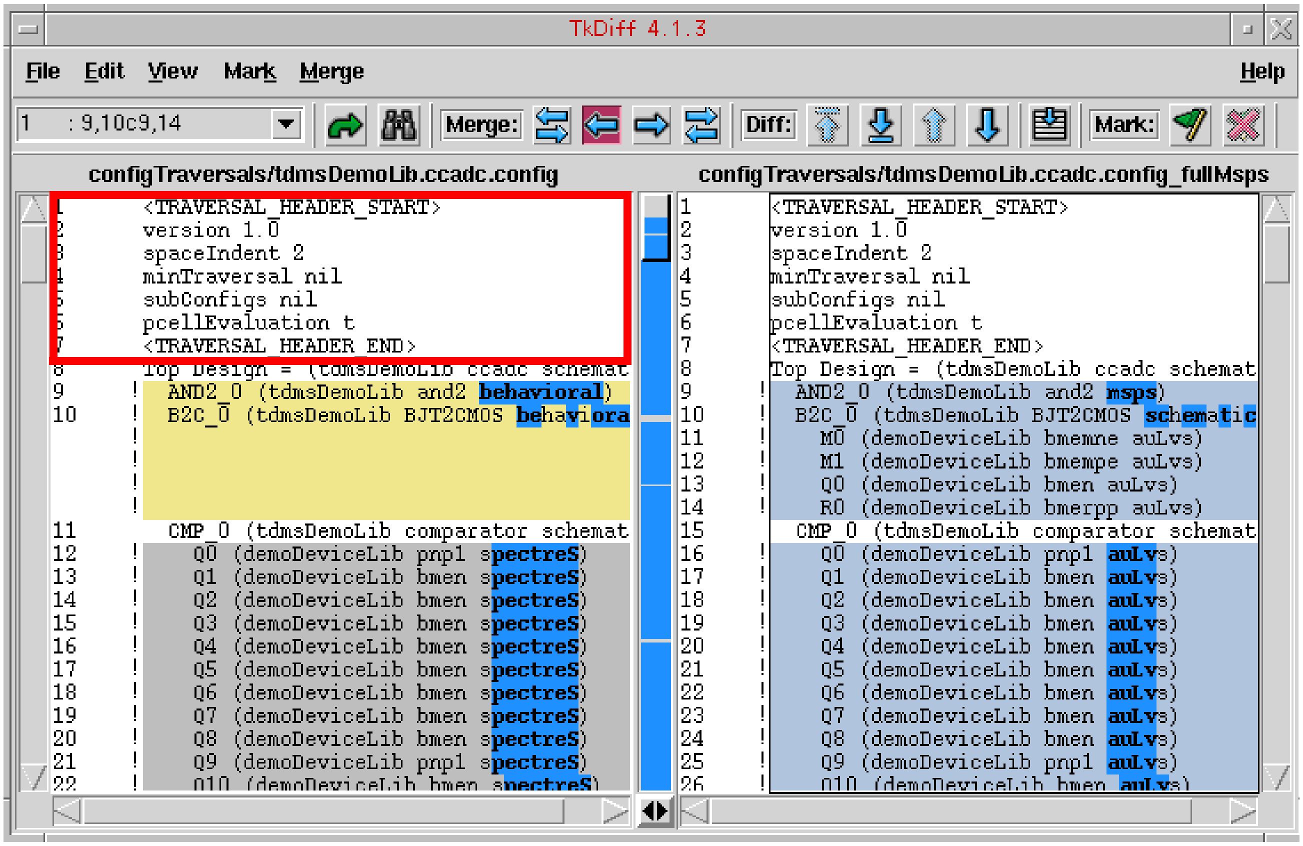
If you use an interactive terminal-based tool for comparison, such as `vimdiff`, you need to open the tool in a new terminal. For example,

```
xterm -e vimdiff
```

- ❑ *Output directory*—Browse and select an output directory in which the traversal results are saved. If you do not specify anything, the `configTraversal` directory, which is present in the directory from where you launch HED, is used to save the traversal files.

- Click *OK* to perform the comparison.
- Click *Apply* to set your selections and leaves the form open.
- Click *Default*s to populate the form fields with the default values from the `hed.env` registry file.

The comparison results are displayed in `tkdiff` as shown in the figure below:



configTraversals/tdmsDemoLib.ccadc.config

```
1           <TRAVERSAL_HEADER_START>
2           version 1.0
3           spaceIndent 2
4           minTraversal nil
5           subConfigs nil
6           pcellEvaluation t
7           <TRAVERSAL_HEADER_END>
8           Top Design = (tdmsDemoLib ccadc schemat
9         !   AND2_0 (tdmsDemoLib and2 behavioral)
10        !   B2C_0 (tdmsDemoLib BJT2CMOS behaviora
          !
          !
          !
          !
11            CMP_0 (tdmsDemoLib comparator schemat
12        !     Q0 (demoDeviceLib pnp1 spectreS)
13        !     Q1 (demoDeviceLib bmen spectreS)
14        !     Q2 (demoDeviceLib bmen spectreS)
15        !     Q3 (demoDeviceLib bmen spectreS)
16        !     Q4 (demoDeviceLib bmen spectreS)
17        !     Q5 (demoDeviceLib bmen spectreS)
18        !     Q6 (demoDeviceLib bmen spectreS)
19        !     Q7 (demoDeviceLib bmen spectreS)
20        !     Q8 (demoDeviceLib bmen spectreS)
21        !     Q9 (demoDeviceLib pnp1 spectreS)
22        !     Q10 (demoDeviceLib bmen spectreS)
```

configTraversals/tdmsDemoLib.ccadc.config_fullMsps

```
1           <TRAVERSAL_HEADER_START>
2           version 1.0
3           spaceIndent 2
4           minTraversal nil
5           subConfigs nil
6           pcellEvaluation t
7           <TRAVERSAL_HEADER_END>
8           Top Design = (tdmsDemoLib ccadc schemat
9         !   AND2_0 (tdmsDemoLib and2 msps)
10        !   B2C_0 (tdmsDemoLib BJT2CMOS schematic
11        !     M0 (demoDeviceLib bmemne auLvs)
12        !     M1 (demoDeviceLib bmempe auLvs)
13        !     Q0 (demoDeviceLib bmen auLvs)
14        !     R0 (demoDeviceLib bmerpp auLvs)
15            CMP_0 (tdmsDemoLib comparator schemat
16        !     Q0 (demoDeviceLib pnp1 auLvs)
17        !     Q1 (demoDeviceLib bmen auLvs)
18        !     Q2 (demoDeviceLib bmen auLvs)
19        !     Q3 (demoDeviceLib bmen auLvs)
20        !     Q4 (demoDeviceLib bmen auLvs)
21        !     Q5 (demoDeviceLib bmen auLvs)
22        !     Q6 (demoDeviceLib bmen auLvs)
23        !     Q7 (demoDeviceLib bmen auLvs)
24        !     Q8 (demoDeviceLib bmen auLvs)
25        !     Q9 (demoDeviceLib pnp1 auLvs)
26        !     Q10 (demoDeviceLib bmen auLvs)
```

The output is displayed in a text format in a graphical tree structure and differences are highlighted.

The traversal results for the same configuration can be different if you specify different values for the options. The options and their values are displayed in the traversal header as highlighted in the figure above.

Using Command-Line for Comparing Configurations

You can also use the command-line interface to traverse and compare two configurations.

The command used to specify the comparison details is as follows:

```
hedConfigCompare [-outdir <dir>] [-s <numspaces>] [-min] [-subcfg]

[-difftool <toolcmd>] [-cdslib <path>] <configSpec1> <configSpec2>
```

where,

- `-outdir <dir>`—Specifies a directory in which the traversal output files are saved. The default directory is `./configTraversals`.
- `-s <numspace>`—Specifies the number of spaces for indentation. The default value is 2 and the value ranges from 1 to 8.
- `-min`—Specifies the minimal traversal, which means traversing through hierarchy by avoiding revisits of the instances that have already been visited.
- `-subcfg`—Specifies whether the traversal results for subconfigurations are included in the output file.
- `-difftool <toolcmd>`—Specifies a tool to be used to display the compare traversal results. if this option is not specified, `tkdiff` is used to display the results.
- `-cdslib <path>`—Specifies the path from where `cdslib` file is to be loaded. If this file is not found, the command does not run.
- `<configSpec1> and <configSpec2>`—These commands are described as follows:

  ```
  {lib.cell:config [-output <outputPath>]}  |  {<inputPath>}
  ```

  - `lib.cell:config`—Specifies an elaborated and traversed configuration view.
  - `-output <outputPath>`—Specifies an output file in which the traversal results are saved. If the output file is not specified, the traversal results are saved in a file in the default output directory.
  - `inputPath`—If you do not want to perform the traversal for the configuration to be compared, you can directly provide the path to the output file.

Examples

- `hedConfigCompare lib1.cell1:config1 lib2.cell2:config2`

  This command traverses both the configurations and save the traversal results in the following output files:

  ```
  ./configTraversals/lib1.cell1.config1
  ```

  ```
  ./configTraversals/lib2.cell2.config2
  ```

  The command then compares these files.

-
  ```
  hedConfigCompare -outdir ./localFiles lib1.cell1:config1 ./
  localFiles/config2.trav
  ```

This command traverses only the first configuration and saves the results in default file in the specified output directory, `./localFiles`. The command then compares `./localFiles/lib1.cell1.config1` and `./localFiles/config2.trav`.

- `hedConfigCompare lib1.cell1:config1 -output ./localFiles/config1.trav  ./localFiles/config2.trav`

  This command traverses only the first configuration, `lib1.cell1:config1`, and saves the traversal results in the specified output file, `./localFiles/config1.trav` in the default output directory (`configTraversals`). This command then compares ./ `localFiles/config1.trav` and `./localFiles/config2.trav`.

- `hedConfigCompare lib1.cell1:config1 -output ./localFiles/config1.trav   lib2.cell2:config2 -output ./localFiles/config2.trav`

  This command traverses both the configurations and saves the results to the following files in the default output directory:

  `./localFiles/config1.trav`

  `./localFiles/config2.trav`

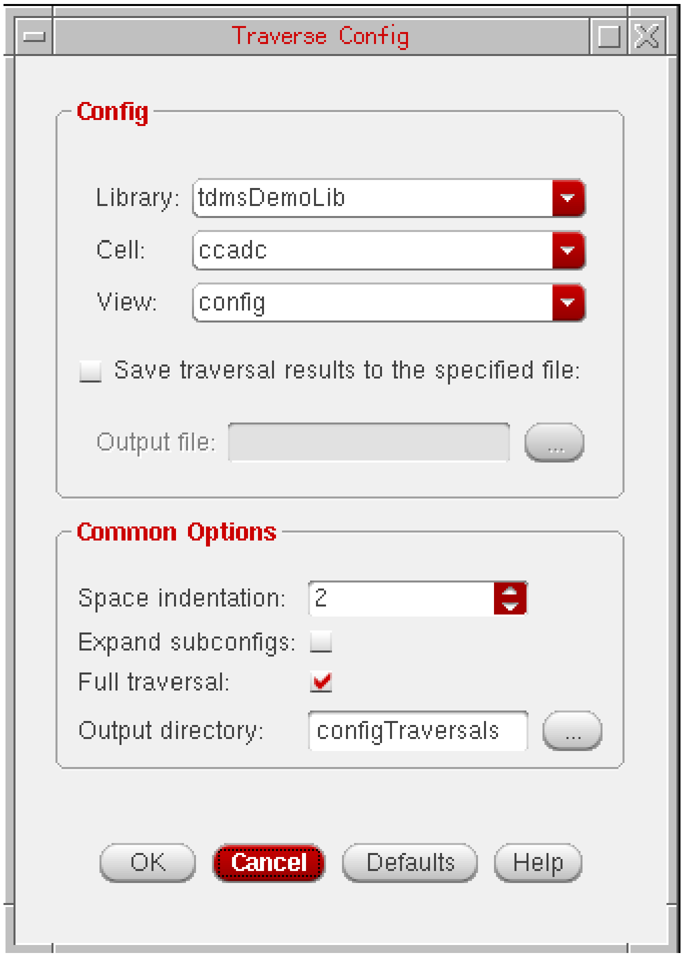
  The command then compares these output files.

Traversing through Configurations

To traverse through a configuration and save the traversal results in an output file, do the following:

➡ Choose *File – Traverse Config*.

The *Traverse Config* form appears as shown in the figure below:



Library: tdmsDemoLib

Cell: ccadc

View: config

□ Save traversal results to the specified file:

Output file:

Space indentation: 2

Expand subconfigs: □

Full traversal: ☑

Output directory: configTraversals

This form includes the following sections:

- In the *Config* section, specify the following fields:
    - ❑ *Library*—Specify the library for the first configuration.
    - ❑ *Cell*—Specify the cell.
    - ❑ *View*—Specify the view.

- ❑ *Save traversal results to the specified file:*—Select this checkbox if you want to specify an output file in which traversal results are to be saved. If you do not select this option, the traversal files are saved in the default traversal output directory named `configTraversals`. The traversal files are saved with the following naming convention:

  `<lib>.<cell>.<view>`

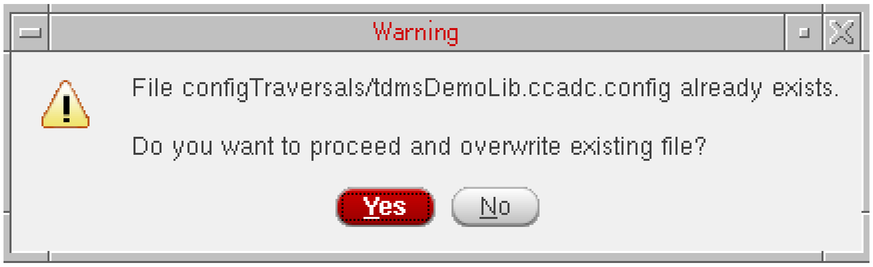
  For example, `tdmsDemoLib.ccadc.config`

  where, `tdmsDemoLib` is a library, `ccadc` is a cell, and `config` is a view.

- ❑ *Output file:* Browse and select the output file in which you want to save the results. If the output file already exists, a warning message appears to confirm overwriting to this file. This option is available only if you select the *Save traversal results to the specified file* checkbox.

- ■ In the *Common Options* section, provide similar options as described in the *Compare Config* form.
- ■ Click *OK* to perform the traversal.
- ■ Click *Defaults* to populate the form fields with the default values from the `hed.env` registration file.

If you save the traversal results into an output file that already has the traversal output, the following warning message appears:



File configTraversals/tdmsDemoLib.ccadc.config already exists.

Do you want to proceed and overwrite existing file?

Using Command-Line for Traversing Configurations

You can also use the command-line interface to traverse the configurations.

The command used to traverse through configuration is as follows:

```
hedConfigTraverse [-outdir <dir>] [-s <numspaces>] [-min] [-subcfg]

[-cdslib <path>] lib.cell:config [-output <outputPath>]
```

where, command options are similar to that of the compare traversal command options.

Using the Virtuoso Schematic Editor with the Hierarchy Editor

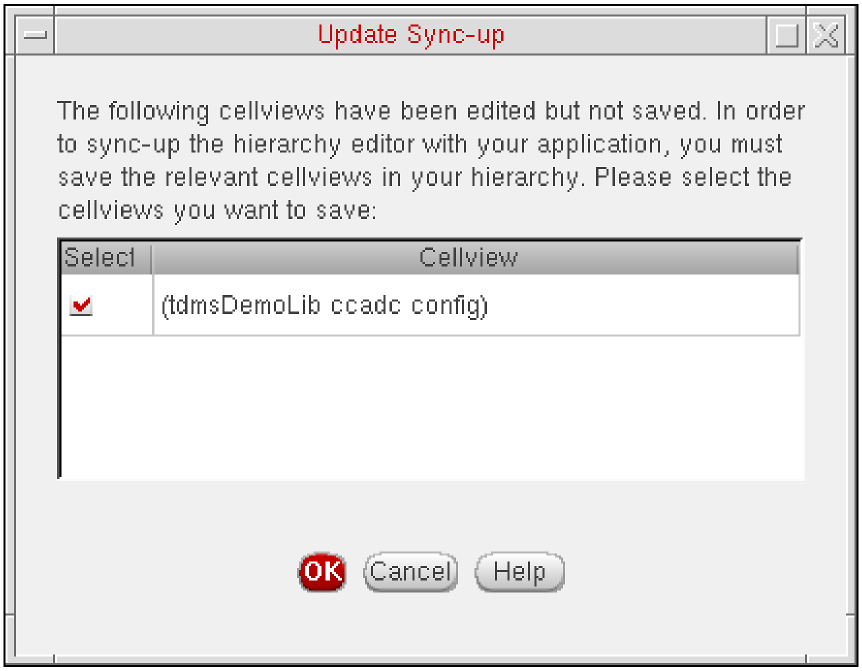
The Hierarchy Editor and the schematic editor can be kept synchronized when they are both started in the Virtuoso design environment. (See “Starting the Hierarchy Editor from the Virtuoso Design Environment” on page 19.)

When you change the configuration in the Hierarchy Editor, the changes are not reflected in the schematic automatically. To update the schematic, you need to do the following:

1. In the Hierarchy Editor, save the configuration.
2. In the Hierarchy Editor, choose *View – Update* to update the configuration.

The configuration is re-read from disk and the schematic displays your changes.

If you do not save the configuration after making your changes, the following form appears when you choose *View – Update* in the Hierarchy Editor:



Update Sync-up

The following cellviews have been edited but not saved. In order to sync-up the hierarchy editor with your application, you must save the relevant cellviews in your hierarchy. Please select the cellviews you want to save:

Select	Cellview
☑	(tdmsDemoLib ccadc config)

In the form,

1. Select the cellviews.
2. Click *OK*.

The configuration is saved and the schematic is also updated.

When you change the schematic, the changes are reflected in the Hierarchy Editor when you update the configuration.

Note: When you start the Hierarchy Editor stand-alone and open a schematic from it, any changes you make to the schematic are not reflected in the Hierarchy Editor even when you update the hierarchy. For the Hierarchy Editor and the schematic editor to be synchronized, you must start them from the Virtuoso design environment.

pc.db Files Update Process for Read-Only Libraries

The Hierarchy Editor (HED) follows the parent-child information of cellviews to traverse a design hierarchy. For text views, this information is found in their `pc.db` files. When the `pc.db` files became out-of-date with respect to the text design files, in IC releases prior to IC6.1.6 ISR12/ICADV12.1 ISR14, you needed to either make the RO library writable (undesirable and/or not permissible), or define a temporary (TMP attribute) location in `cds.lib`.

To simplify this model, a new automatic mechanism has been introduced that does not require you to intervene in the `pc.db` file update process. The `pc.db` file is automatically created and saved to the `$CWD/.pcdb` directory when HED detects that it is either missing or out-of-date. HED finds and reads the `pc.db` file from this cache location. The generated `pc.db` files are also copied to the library area, if the library is writable.

Tip

It is not recommended, but you can change the location of `.pcdb` directory by setting the following environment variable.

```
setenv PCDB_CACHE_DIR <writable directory>
```

If you are currently using explicit TMP location in `cds.lib`, the following sequence is followed:

1. HED reads the `pc.db` files from the cache location.
2. If the `pc.db` files are not found in the cache location, HED reads the `pc.db` files from the explicit TMP location.
3. If the `pc.db` files are not found in the explicit TMP location, HED reads the `pc.db` files from the library area.

Note: The explicit TMP is becoming obsolete and setting the TMP attribute in any `cds.lib` file is not recommended anymore for RO libraries.

If `pc.db` files are maintained as golden data in the RO libraries and explicit TMP is not being used in `cds.lib`, the new capability will make the whole flow —from configuration to netlist to simulation— to work automatically in the rare case when `pc.db` files become out-of-date.

Note: The tool currently issues an error when it detects that a `pc.db` file is out-of-date and cannot update this `pc.db` in a RO directory.

About Snapshot Config

The Snapshot Config feature provides a way to freeze a certain config where all cells are explicitly bound in the whole design hierarchy. When this frozen config is reloaded, changes on a disk do not affect the binding when the library and view list are applied .

The Snapshot Config is served just like a generic config but with all the cells explicitly bound. It can be created, saved, edited, and loaded like a generic config. It is not supported in standalone HED.

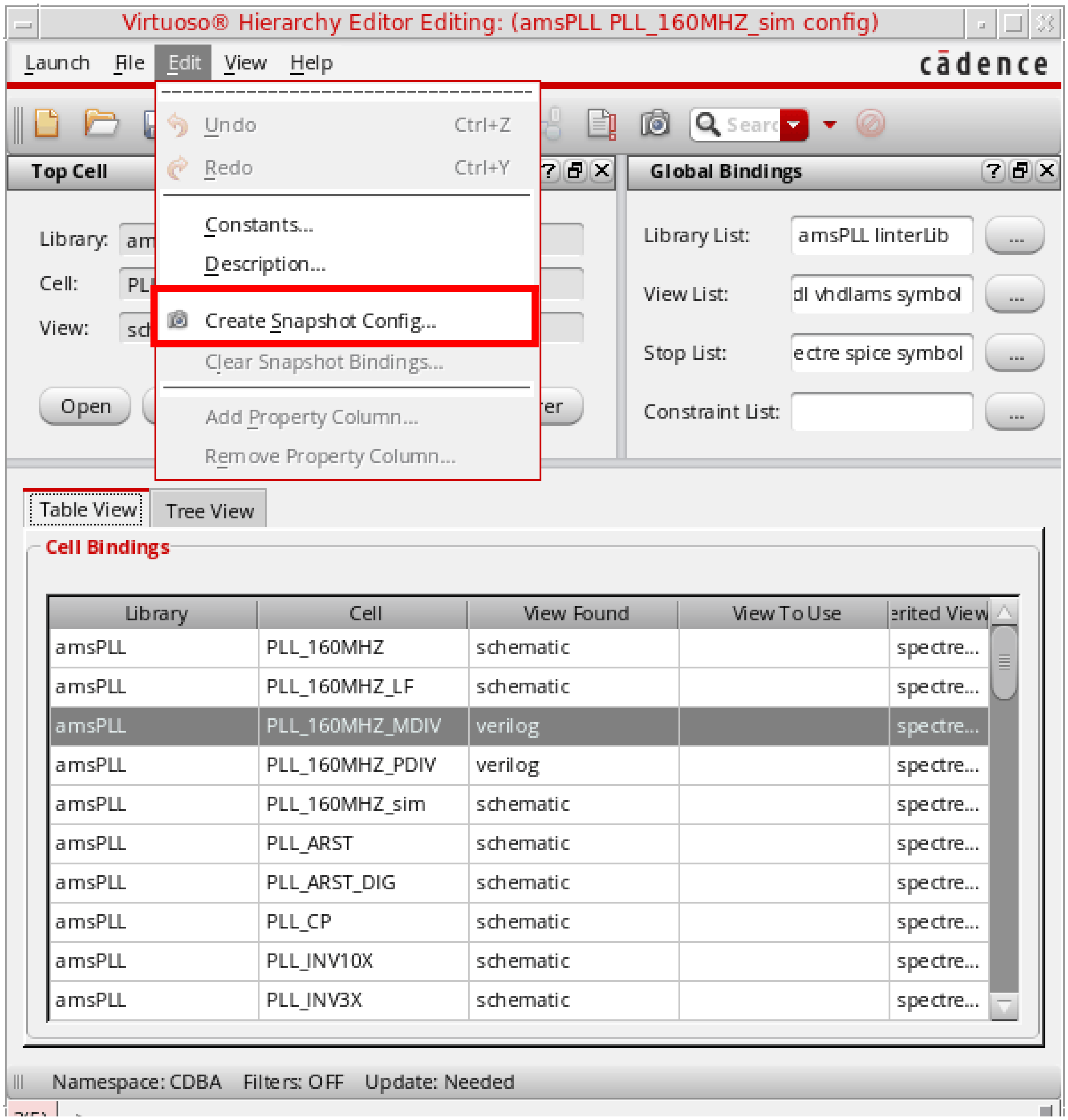
Creating a Snapshot Config

Important

> Before creating a Snapshot Config, you need to save changes (if any) to the original config.

To create Snapshot Config, you need to perform the following steps:

1. From the Virtuoso Hierarchy Editor window, select *Edit – Create Snapshot Config*. Otherwise, you can select the *Create a snapshot config* icon (📷) on the HED toolbar.



Library: am
Cell: PL
View: sc

Library List: amsPLL linterLib
View List: dl vhdlams symbol
Stop List: ectre spice symbol
Constraint List:

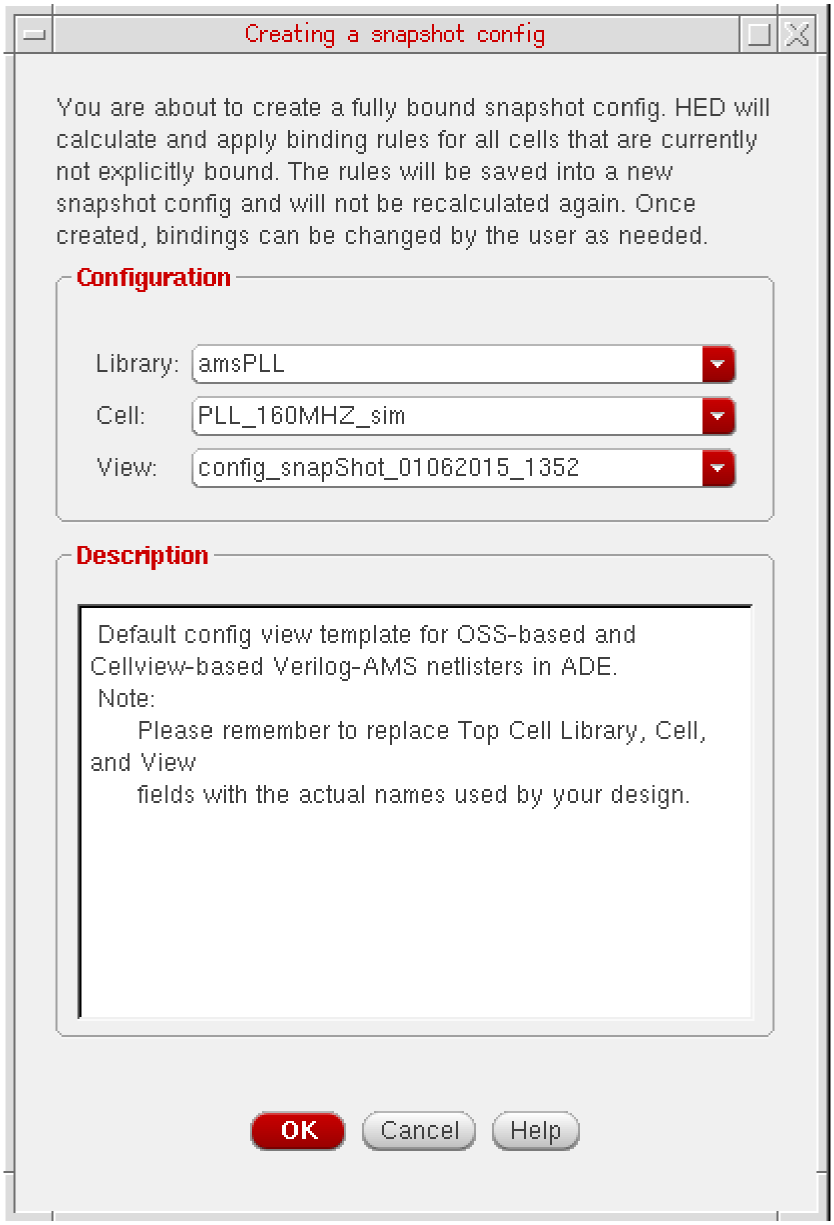
Cell Bindings

Library	Cell	View Found	View To Use	erited View
amsPLL	PLL_160MHZ	schematic		spectre...
amsPLL	PLL_160MHZ_LF	schematic		spectre...
amsPLL	PLL_160MHZ_MDIV	verilog		spectre...
amsPLL	PLL_160MHZ_PDIV	verilog		spectre...
amsPLL	PLL_160MHZ_sim	schematic		spectre...
amsPLL	PLL_ARST	schematic		spectre...
amsPLL	PLL_ARST_DIG	schematic		spectre...
amsPLL	PLL_CP	schematic		spectre...
amsPLL	PLL_INV10X	schematic		spectre...
amsPLL	PLL_INV3X	schematic		spectre...

Namespace: CDBA Filters: OFF Update: Needed

2. The Creating a snapshot config dialog box is displayed. The default format of snapshot name is `<config_name>_snapShot_<date>_<time>`.

   In the *Description* section, the information is carried over from the original configuration. You can update the view name and description as required.



Creating a snapshot config

You are about to create a fully bound snapshot config. HED will calculate and apply binding rules for all cells that are currently not explicitly bound. The rules will be saved into a new snapshot config and will not be recalculated again. Once created, bindings can be changed by the user as needed.

Configuration

Library: amsPLL

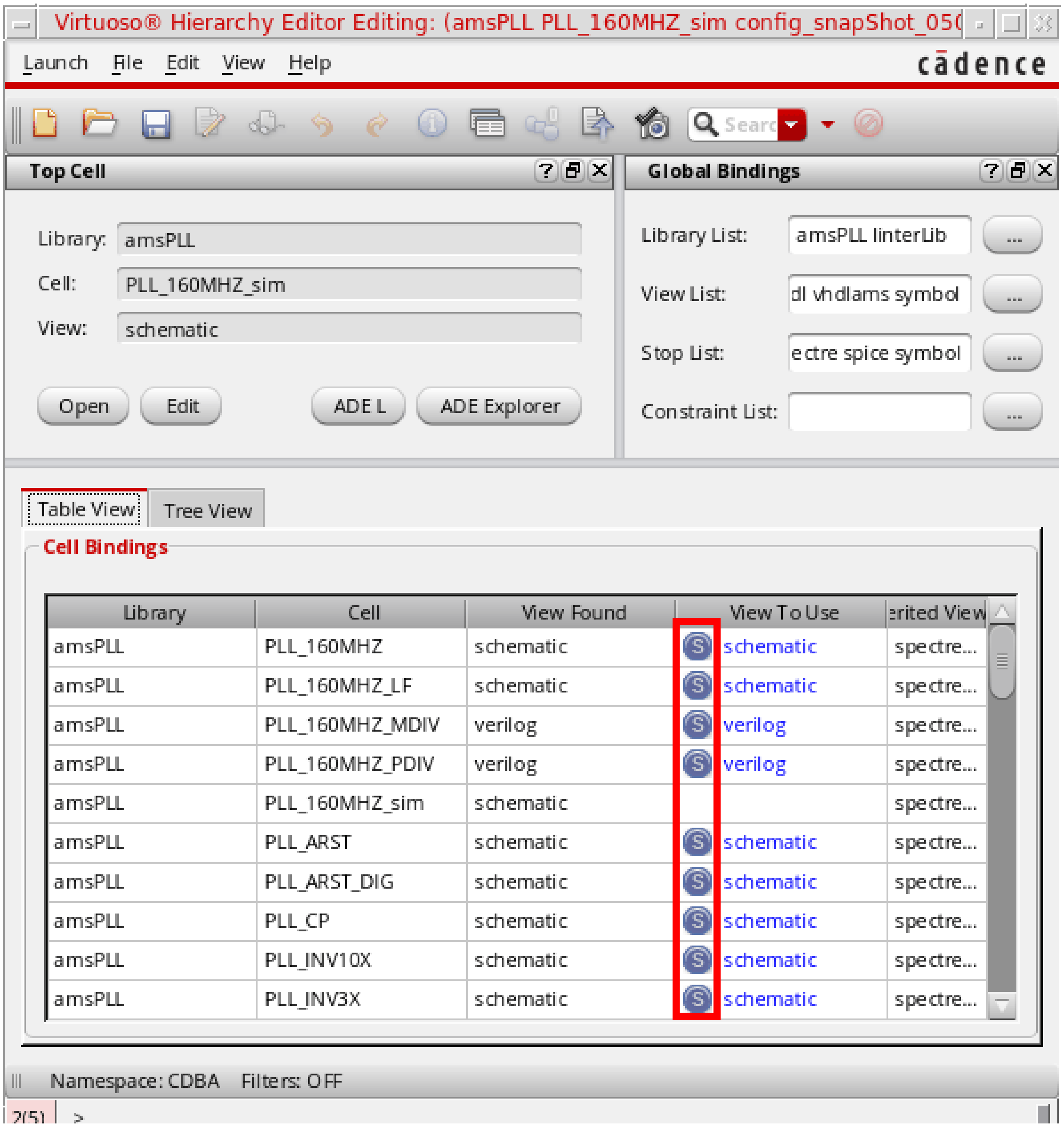
Cell: PLL_160MHZ_sim

View: config_snapShot_01062015_1352

Description

```
 Default config view template for OSS-based and
Cellview-based Verilog-AMS netlisters in ADE.
 Note:
      Please remember to replace Top Cell Library, Cell,
and View
      fields with the actual names used by your design.
```

3. Click the *OK* button, to create a fully bound snapshot config. As highlighted below, the snapshot binding are shown with the “S” symbol.


Virtuoso® Hierarchy Editor Editing: (amsPLL PLL_160MHZ_sim config_snapShot_05(
Launch File Edit View Help
cādence
Searc
Top Cell
Global Bindings
Open
Edit
ADE L
ADE Explorer
Table View
Tree View
Cell Bindings
2(5)
>

Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

Library List: amsPLL linterLib
View List: dl vhdlams symbol
Stop List: ectre spice symbol
Constraint List:

Library	Cell	View Found	View To Use	erited View
amsPLL	PLL_160MHZ	schematic	Ⓢ schematic	spectre...
amsPLL	PLL_160MHZ_LF	schematic	Ⓢ schematic	spectre...
amsPLL	PLL_160MHZ_MDIV	verilog	Ⓢ verilog	spectre...
amsPLL	PLL_160MHZ_PDIV	verilog	Ⓢ verilog	spectre...
amsPLL	PLL_160MHZ_sim	schematic		spectre...
amsPLL	PLL_ARST	schematic	Ⓢ schematic	spectre...
amsPLL	PLL_ARST_DIG	schematic	Ⓢ schematic	spectre...
amsPLL	PLL_CP	schematic	Ⓢ schematic	spectre...
amsPLL	PLL_INV10X	schematic	Ⓢ schematic	spectre...
amsPLL	PLL_INV3X	schematic	Ⓢ schematic	spectre...

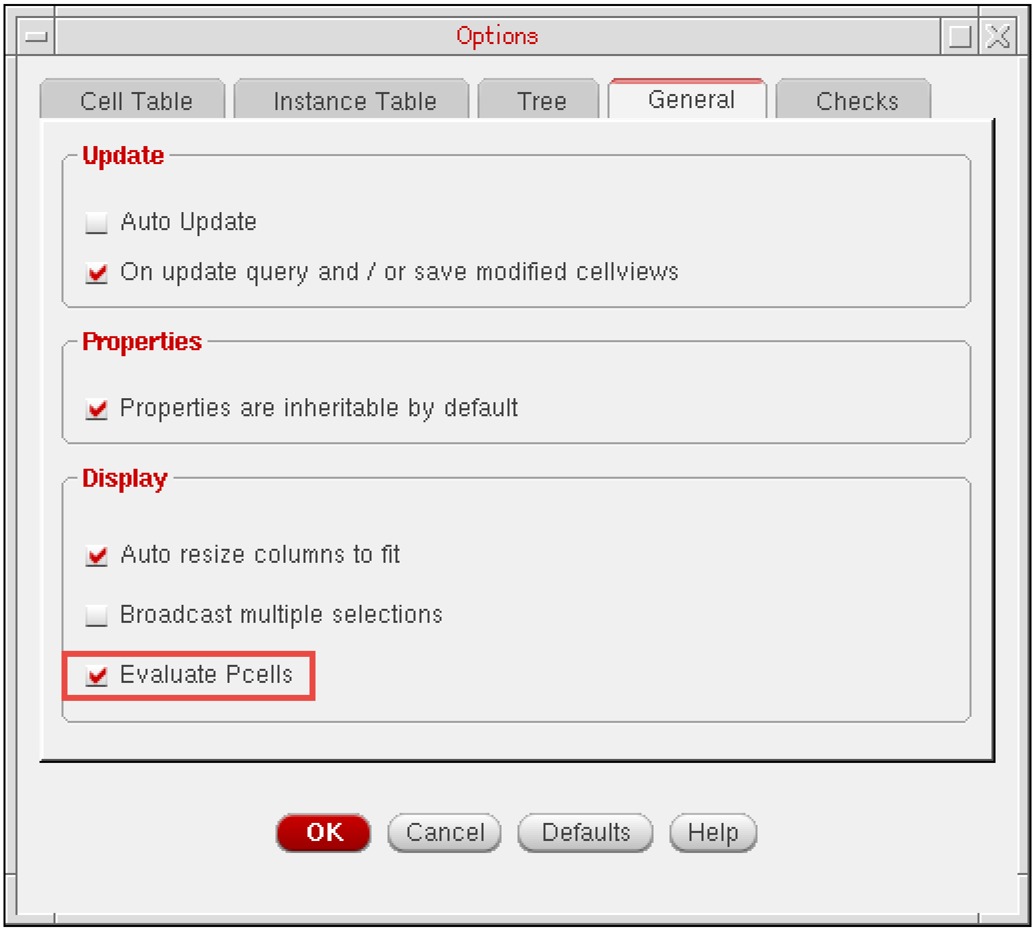
Namespace: CDBA Filters: OFF

Note: The cells or instances with “S” symbol imply the binding is set by the snapshot feature, whereas, those without “S” imply that it is explicitly set. The top design is `"amsPLL" "PLL_160MHZ_sim" "schematic"` and does not have an explicit binding set by the create snapshot functionality.

Important

To get an accurate snapshot config for a design containing Pcells, open the Options form by selecting *View – Options* from the Virtuoso Hierarchy Editor window. Then, in the *General* tab, select the *Evaluate Pcells* check box, as highlighted below.

If the Pcell evaluation is turned off, a warning message will be displayed when the snapshot is created or checked.



Update

☐ Auto Update

☑ On update query and / or save modified cellviews

Properties

☑ Properties are inheritable by default

Display

☑ Auto resize columns to fit

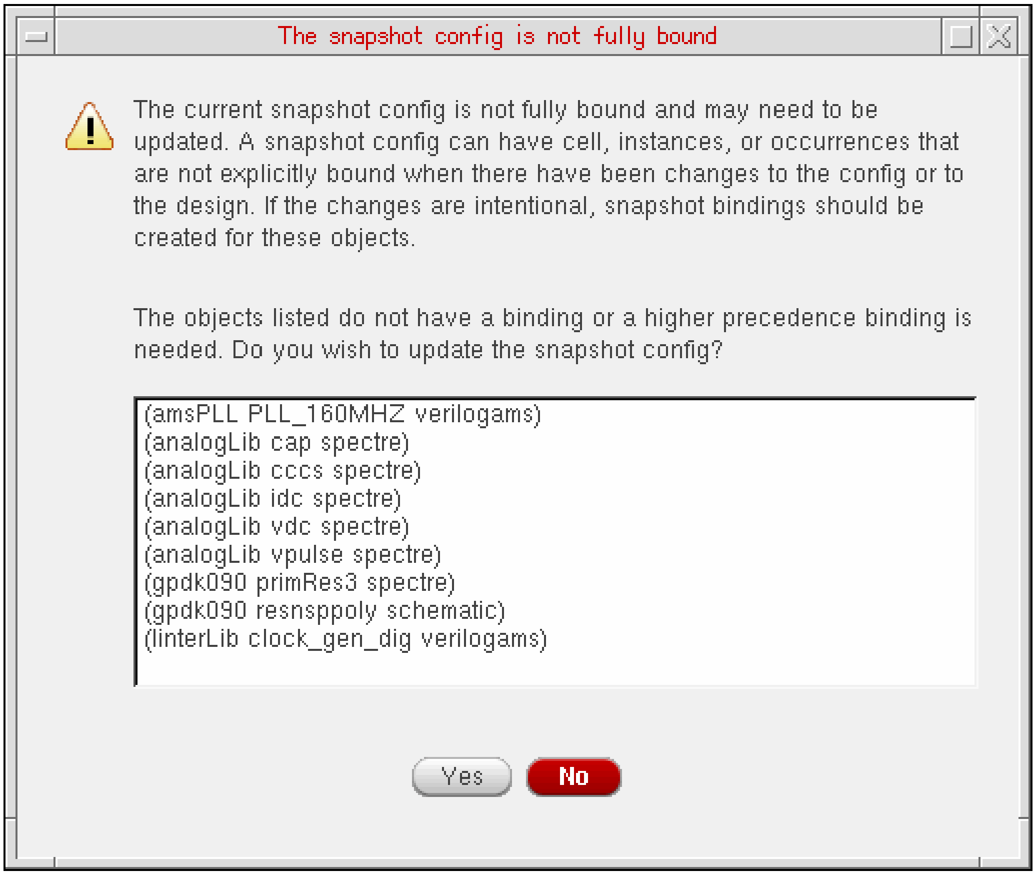
☐ Broadcast multiple selections

☑ Evaluate Pcells

Checking a Snapshot Config

This option manually checks for the cells and instances that do not have an explicit binding. While updating the snapshot, if a cell is not explicitly bound with the right precedence level, then it will get updated with the same flow used to the set bindings when creating a snapshot.

For example, if the snapshot config is not fully bound then the following warning message is displayed.



The snapshot config is not fully bound

The current snapshot config is not fully bound and may need to be updated. A snapshot config can have cell, instances, or occurrences that are not explicitly bound when there have been changes to the config or to the design. If the changes are intentional, snapshot bindings should be created for these objects.

The objects listed do not have a binding or a higher precedence binding is needed. Do you wish to update the snapshot config?

```
(amsPLL PLL_160MHZ verilogams)
(analogLib cap spectre)
(analogLib cccs spectre)
(analogLib idc spectre)
(analogLib vdc spectre)
(analogLib vpulse spectre)
(gpdk090 primRes3 spectre)
(gpdk090 resnsppoly schematic)
(linterLib clock_gen_dig verilogams)
```

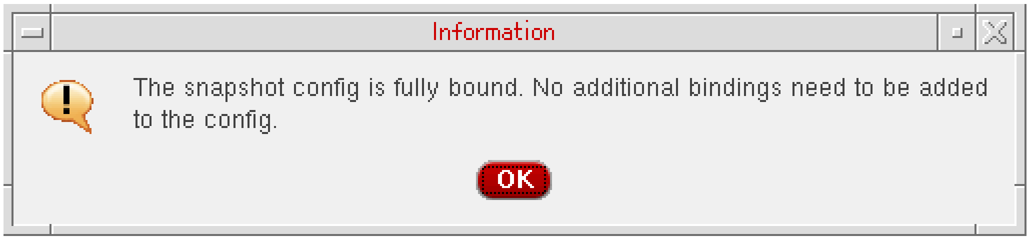
To add binding for the objects listed in the dialog box, click the *Yes* button.

Caution

Any invalid cellviews or unused bindings should be corrected prior to updating a snapshot.

To check the snapshot config, select *Edit – Check Snapshot Config* from the Virtuoso Hierarchy Editor window. Otherwise, you can also select the *Check a snapshot config* icon (✓) on the HED toolbar.

In case the snapshot config is fully bound, the following dialog box is displayed.



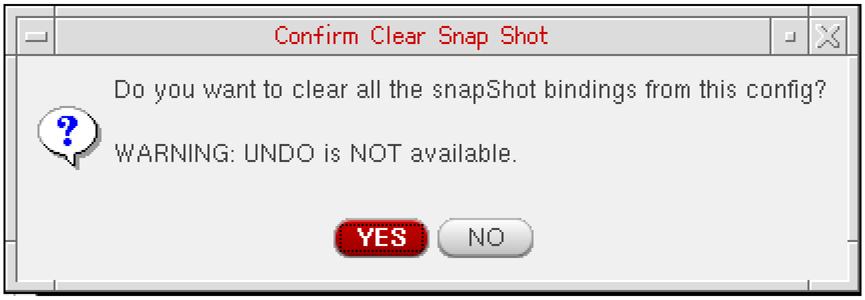
The snapshot config is fully bound. No additional bindings need to be added to the config.

Note: HED does not create snapshot bindings for sub-configs. Ideally, you should first create the snapshot sub-configs and then use these snapshot sub-configs in the top-level config. A warning message will be displayed if non-snapshot sub-configs are being used by a top-level snapshot config.

Clearing Snapshot Bindings

This option is used to remove all snapshot bindings. To clear the snapshot bindings:

1. Select *Edit – Clear Snapshot Binding* from the Virtuoso® Hierarchy Editor window. The Confirm Clear Snap Shot dialog box is displayed.



Do you want to clear all the snapShot bindings from this config?

WARNING: UNDO is NOT available.

2. . Click *YES* to clear the snapshot bindings.

3

Changing Design Components

This chapter covers the following topics:

Overview of Configuration Rules

With the Hierarchy Editor, you can control the expansion of your design for any given purpose such as simulation or netlisting. You do this by creating and editing rules that define a design configuration. You can specify the following configuration rules in the Hierarchy Editor:

- **Library Binding:** Library binding determines the library from which a cell is obtained.
- **View Binding:** View binding determines which view of a cell is to be used in the configuration. For example, a cell might have different representations such as schematic, Verilog, and VHDL; the view binding determines which of these representations is used in the configuration.
- **Stop Lists and Stop Points**: Stop lists and stop points prevent further expansion of any part of a design.
- **Bind-to-Open:** The bind-to-open attribute is a way of specifying that a single instantiation of a cell is to be skipped by setting a bind-to-open attribute on it.
- **Constraint Binding**: Constraint binding determines which constraint view is to be used in the configuration.

You create and edit these rules at different levels of the design. You can specify global rules that apply to every level of the design. You can also specify rules at the cell, instance, and occurrence levels.

Note: Constraint binding can only be defined at the global level.

Global Bindings

Global bindings are configuration rules that are defined at the global level. See also Global Bindings Section on page 46.

At the global level, you can define a library list, view list, and stop list. These global bindings become the default rules for the configuration and apply to every level of the hierarchy, unless they are overridden at lower levels of the hierarchy.

For more information about global bindings, see “Defining Rules at the Global Level” on page 148.

Cell Bindings

Cell bindings are configuration rules that are defined at the cell level. Cell bindings override global bindings for a cell.

Cell bindings apply to all instantiations of a cell.

You can define the following rules at the cell level:

- View binding that determines which view of the cell is used in the configuration

  You can define view bindings for a cell in the following ways:

  - ❑ Binding rules for a specific cell that apply to that cell and to components below it in the hierarchy, until they are overridden by another binding
  - ❑ Binding for a specific cell that applies only to that cell and is not inherited by components below the cell
- Library binding that determines the library from which the cell is obtained. This is done by specifying a library list for the cell. Cell-level library lists are inherited by components below the cell in the hierarchy.
- Stop point on the cell that prevents the cell from being expanded.
- Bind-to-open attribute specifies that a single instantiation of a cell is to be skipped.

For more information about cell bindings, see “Defining Rules at the Cell Level” on page 154.

Instance Bindings

Instance bindings are configuration rules that are defined at the instance level. Instance bindings override global bindings and cell bindings for an instance.

Instance bindings apply to a single instantiation of a cell. Note, however, that if the cell that contains the instance is used in multiple places in the design, the binding applies to the instance in all those places. If you want to specify a binding for only one instance at a specific path, you need to specify occurrence bindings instead of instance bindings.

You can define the following rules at the instance level:

- View binding that determines which view of the cell is used in the configuration

  You can define view binding for an instance in the following ways:

  - ❑ Binding rules for a specific instance that apply to that instance and to components below it in the hierarchy, until they are overridden by another binding
  - ❑ Binding for a specific instance that applies only to that instance and is not inherited by components below it in the hierarchy

- Library binding that determines the library from which the instance is obtained. This is done by specifying a library list for the instance. Instance-level library lists are inherited by components below the instance in the hierarchy.
- Stop point on the instance that prevents the instance from being expanded.
- Bind-to-open attribute that specifies that the instance is unbound, that is, it is not bound to a particular library, cell, or view.

For more information about instance bindings, see "Defining Rules at the Instance Level" on page 159.

Occurrence Bindings

Occurrence bindings are configuration rules that are defined at the occurrence level. An occurrence is an instance that is defined by the full path from the top-level design to the instance. Therefore, occurrence bindings apply to a single object at a specific path in the design.

You can specify the following rules at the occurrence level:

- Library, cell, and view binding for a single object at a specific path in the design
- Stop point that prevents the occurrence from being expanded
- Bind-to-open attribute that specifies that the occurrence is unbound, that is, it is not bound to a specific library, cell, or view.

For more information about occurrence bindings, see "Defining Rules at the Occurrence Level" on page 168.

Note: Not all Cadence tools support occurrences. Refer to the documentation for the applications you are using to check if those applications support occurrences.

How a Library Is Selected for an Object

You can define library binding rules at different levels of a design. The Hierarchy Editor uses the following order of precedence (listed from highest precedence to lowest precedence) to select the library for an object:

1. Occurrence binding
2. Instance-level Inherited Library List
3. Cell-level Inherited Library List

4. Global Library List

You can see which binding rules were used to select the library for an object by reading the description in the Explain form in the Hierarchy Editor. See “Verifying Configuration Data” on page 98 for more information.

How a View Is Selected for an Object

You can define view binding rules in different ways and at different levels of a design. The Hierarchy Editor uses the following order of precedence (listed from highest precedence to lowest precedence) to select the view for an object:

1. Occurrence binding
2. Instance-level View to Use binding
3. Cell-level View to Use binding
4. Instance-level Inherited View List
5. Cell-level Inherited View List
6. Global View List

You can see which binding rules were used to select the view for an object by reading the description in the Explain form in the Hierarchy Editor. See “Verifying Configuration Data” on page 98 for more information.

Defining Rules at the Global Level

You can define rules at the global level that become the default rules for the entire configuration. These rules are inherited by every level of the hierarchy until they are overridden by rules lower in the hierarchy.

You specify global rules, referred to as global bindings, in the *Global Bindings* section of the Hierarchy Editor (see Global Bindings Section and View List Building Forms). You can specify these rules in either the table view or the tree view of the Hierarchy Editor.

You can specify the following rules at the global level:

- Library List
- View List
- Stop List

- Constraint List

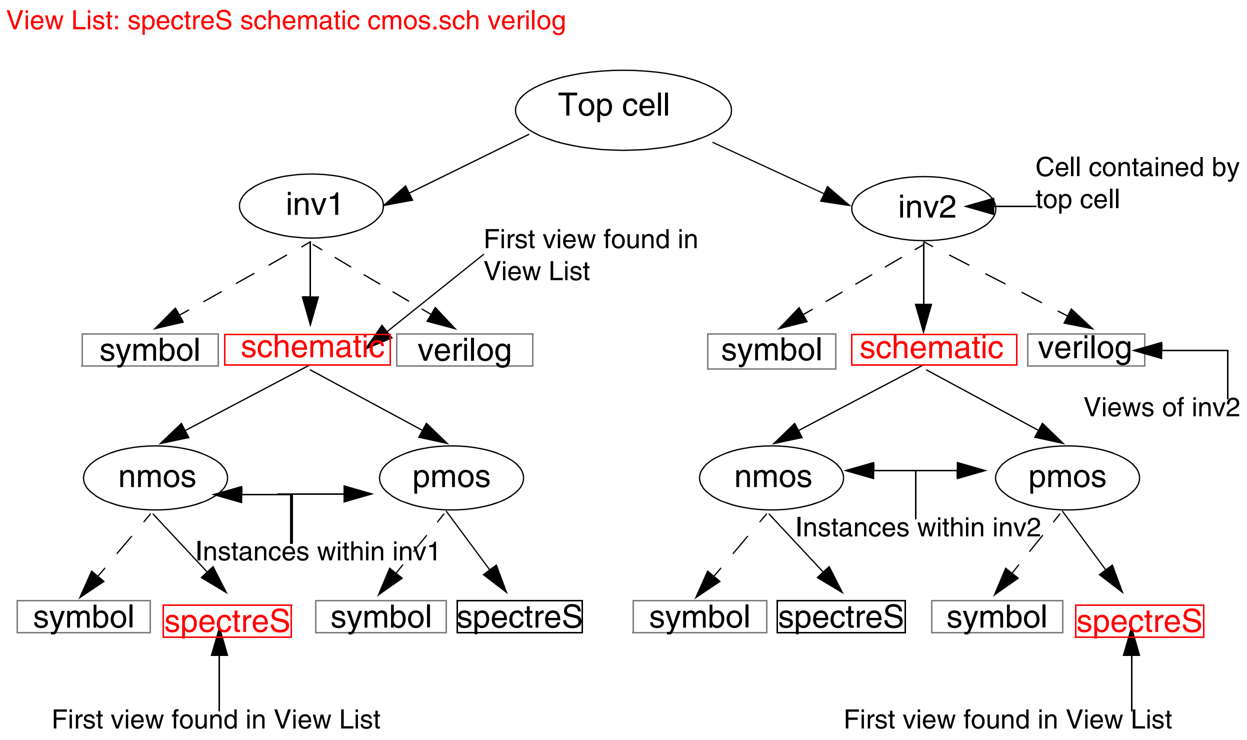
Defining a View List at the Global Level

When you define views at the global level, the views apply to every level of the configuration and must be listed in the order in which you want them searched.

For example, if your View List is

```
spectreS schematic cmos.sch verilog
```

the Hierarchy Editor searches for each instance as shown in the figure.



The figure shows how the Hierarchy Editor searches for the first view listed in the View List. For example, in the cell `inv1`, the `symbol`, `schematic`, and `verilog` views are in the table. If the Hierarchy Editor finds a match, it selects that view for the design. If it does not find a match, it continues to look until a match is found. In the example, the second view in the view list, `schematic`, is the first view that is found.

The View List determines which view is selected for every object in the design, unless overridden by a cell binding, instance binding, or occurrence binding.

If the Hierarchy Editor does not find a view for a cell or instance, it displays `**NONE**` for that cell or instance in the *View Found* column of the *Cell* or *Instance Bindings* table. `**NONE**` indicates a binding error that prevents you from netlisting until you correct the error.

The global view list that you define becomes the default Inherited View List for each cell and instance. You can override the default at the cell, instance, or occurrence levels. You can also check which binding rules were used to select the view for an object; see "Verifying Configuration Data" on page 98 for more information.

To create or change the global View List,

1. In the *Global Bindings* section*,* click in the *View List* field.
2. Type the views you want in the order of preference. Separate the entries with a space (see also View List Building Forms on page 48).

   You can also define a constant, such as `$default`, `$digital`, or `$analog,` to represent a list of views and use this constant in the view list. Constants are easier to read than a long view list, and they can be re-used in any view list. See "Using Constants" on page 189 for more information.

   You can also use the asterisk character (`*`) as a wildcard in the view list; see "Using Wildcards in a View List" on page 193 for more information.
3. Press `Return`.
4. To see the changes in the configuration, click the *Update* icon in the toolbar.

   The *Tree View* and *Table View* change to show the new configuration resulting from the change in the View List.

   If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.
5. To save the changes, from the menu bar, choose *File – Save or* press `Control-s.`

To delete a view from the global View List,

1. In the *Global Bindings* section*,* click in the *View List* field.
2. Highlight the view you no longer want and press `Delete`.
3. To see the changes in the configuration, click the *Update* icon in the toolbar.

Defining a Library List at the Global Level

Library lists determine the library (assuming that the library is not fixed by, for example Virtuoso) from which each component in the design is obtained. Libraries are listed in order of preference. The Hierarchy Editor searches the libraries in the order in which they are listed and uses the component from the first library in which it is found.

Global library lists apply to every level of the configuration. Global library lists can be overridden by library lists at the cell and instance levels and by specific library.cell:view binding on an occurrence. You can see which binding rules were used to select a library for a component; see "Verifying Configuration Data" on page 98 for more information.

To create or change a library list at the global level,

1. In the *Global Bindings* section of the Hierarchy Editor, click in the *Library List* field.
2. Do one of the following:
   - ❑ To create a new library list, type the libraries in the order of preference or use one of the View List Building Forms.
   - ❑ To add a library to an existing library list, type the new library in the appropriate place. The libraries are searched in the order in which they are listed.
   - ❑ To remove a library from the library list, highlight the library and press `Delete`.
3. Press `Return`.
4. To see the changes in the configuration, click the *Update* icon in the toolbar.

   If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.
5. To save the changes, choose *File – Save* or press `Control-s`.

Defining a Stop List at the Global Level

A stop list is a list of views that are to be treated as if they are at the leaf level, that is they are to be treated as if they do not have any levels of hierarchy below them and cannot be expanded further. The stop list is optional—the Hierarchy Editor does not require you to have a stop list in your configuration.

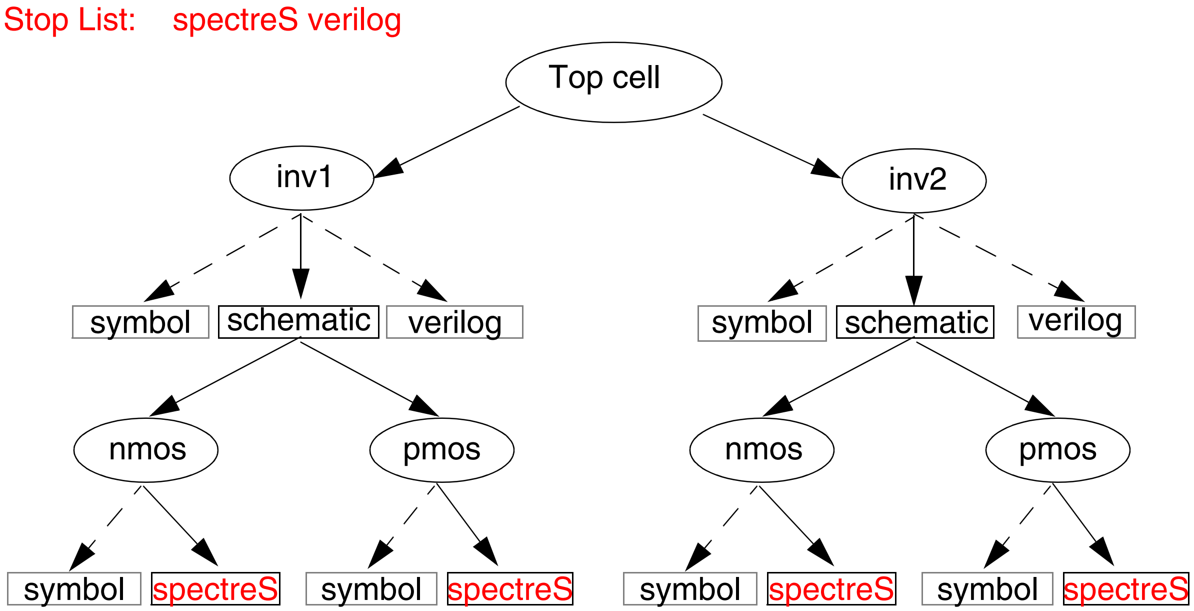
You can use the stop list to designate cellviews that contain no instances. In addition, you can use the stop list to indicate cases where you do not want the configuration to descend further into the hierarchy.

For example, in flattening for placement, you can use the stop list to indicate a cellview that is preplaced, even though it contains instances, so that the placement tool will leave it intact.

A stop list at the global level applies to every level of the configuration. Views must be listed in order of preference. For example, if your stop list is

```
spectreS verilog
```

the Hierarchy Editor expands the design as shown in the figure below.



To create or change the Stop List,

1. In the *Global Bindings* section, click in the *Stop List* field.
2. Do one of the following:
   - ❑ To create a new stop list, type the views in the order of preference (see also View List Building Forms).
   - ❑ To add a view to an existing stop list, type the new view in the appropriate place. The views are searched in the order in which they are listed.
   - ❑ To remove a view from the stop list, highlight the view and press `Delete`.
3. Press `Return`.
4. To see the changes in the configuration, click the *Update* icon in the toolbar.

   If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

The *Tree View* and *Table View* change to show the new configuration resulting from the change in the Stop List.

5. To save the changes, choose *File – Save* or press `Control-s`.

Defining a Constraint List at the Global Level

The global constraint list determines the constraints that are used for the configuration. Constraints are design rules that are to be followed during physical implementation. Constraints are stored in views.

The constraint list is a list of constraint views, listed in order of preference. The first view that is found is used. If none of the constraint views in the list are found, the schematic editor creates an empty view named after the first view in the constraint list.

A constraint list can only be defined at the global level.

To create or change a constraint list at the global level,

1. In the *Global Bindings* section of the Hierarchy Editor, click in the *Constraint List* field.
2. Do one of the following:
   - ❑ To create a new constraint list, type the constraint views in order of preference (see also View List Building Forms).
   - ❑ To add a constraint view to an existing constraint list, type the new view name in the appropriate place. The views are searched for in the order in which they are listed.
   - ❑ To remove a constraint view from the constraint list, select the view name and press `Delete`.
3. Press `Return`.
4. To see the changes in the configuration, click the *Update* icon in the toolbar.

   If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.
5. To save the changes, choose *File – Save* or press `Control-s`.

Note: If you want to specify constraints for a cell, instance, or occurrence, you can create a sub-configuration for that object and specify a constraint list for the sub-configuration. See the *Propagation of Constraints Between Schematics and Layout* section in the *Virtuoso Unified Custom Constraints User Guide* for more information.

For more information about constraints, see the *Virtuoso Unified Custom Constraints User Guide*.

Defining Rules at the Cell Level

You can define rules at the cell level that override the global rules for an individual cell. These rules apply to all instantiations of the cell. In addition, some of these rules can also apply to the hierarchy below the cell.

You specify cell-level rules, also referred to as cell bindings, in the *Cell Bindings* section of the Hierarchy Editor. You must specify cell binding rules in the table view of the Hierarchy Editor; you cannot specify them in the tree view.

You can specify the following rules at the cell level:

- Library Binding
- View Binding
- Stop Point
- Bind-to-Open

Note: You can save cell binding information to a text file. See “Saving Cell Bindings Table Data to a Text File” on page 195 for more information.

Changing Library Bindings on a Per-Cell Basis

You can define a cell-level library list that overrides the global library list for an individual cell. The library list determines the library from which the specific cell is obtained. Libraries are listed in the order of preference. The Hierarchy Editor searches the libraries in the order in which they are listed and uses the cell from the first library in which it is found.

The cell-level library list:

- Applies to every instantiation of the cell
- Is inherited by all components in the hierarchy below the cell

Cell-level library lists can be overridden at the instance and occurrence levels.

To create a cell-level library list,

1. Choose *View – Parts Table*.

2. In the *Cell Bindings* section, click in the *Inherited Lib List* column of the cell whose binding you want to change.

   The field becomes editable. If the field does not become editable, the library is fixed for your design and cannot be changed.

3. Edit the inherited library list. The library list that is currently displayed is the global library list. Type the libraries you want to use in the order of preference.

4. Press `Return`.

5. To view the changes in the configuration, click the *Update* icon.

   If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

6. To save the changes, choose *File – Save* or press `Control-s`.

In the *Instance Bindings* table, you can see the changes applied to the instances the cell contains. If an instance has an instance-level library list, the cell-level list does not apply to that instance.

Changing View Bindings on a Per-Cell Basis

You can define a cell-level view binding that overrides the global view list for an individual cell. Cell-level view binding applies to every instantiation of the cell.

You can either change the binding for only the cell you specify or you can change it in a way that it is also inherited by the components below the cell in the hierarchy. To make the cell binding apply only to the cell, you specify the view in the *View to Use* column. To make the cell binding applicable to components below the cell, you specify a view or view list in the *Inherited View List* column.

To specify a cell binding that applies to the cell as well as to objects below it in the hierarchy,

1. In the *Inherited View List* field, type the new view or list of views for the cell and press `Return`.

   You can use constants in the view list; see “Using Constants” on page 189 for more information. You can also use the asterisk character (`*`) as a wildcard in the view list; see “Using Wildcards in a View List” on page 193 for more information.

2. To see the results of your changes, click the *Update* icon in the toolbar or, from the menu bar, choose *View – Update.*

   If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

The view list you entered is now the view list used for that cell and for any other cells below that cell in the hierarchy (unless they are overridden by other cell or instance bindings lower in the hierarchy).

3. To save the changes, from the menu bar, click *File – Save* or press `Control-s`.

To change a binding for a cell on a per cell basis,

1. In the *Cell Bindings* table, select the cell whose bindings you want to change.
2. Do one of the following:
   - ❑ Right-click the cell and, from the pop-up menu that appears, select *Set Cell View – `newViewName`*.

     Note: The *Source File* and *Reference Verilog* options on the list of available views let you specify a text file. See "Using Text Files in Your Configuration" on page 180 for more information.
   - ❑ Click in the *View to Use* column, type the new view for the cell, and press `Return`.

     The new view name appears in the *View to Use* column in the color used to display user bindings.
   - ❑ To remove a binding, right-click the cell and, from the pop-up menu that appears, select *Set Cell View – <none>*.
3. To see the results of your changes, click the *Update* icon in the toolbar.

   If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.
4. To save the changes, from the menu bar, click *File – Save* or press `Control-s`.

Note: This new cell binding is not inheritable.

Defining Stop Points on a Per-Cell Basis

In addition to a global stop list that applies to the entire configuration, you can specify stop points on individual cells. A stop point on a cell prevents the cell from being expanded when the hierarchy is expanded.

Note: A stop point on a cell applies to all the instantiations of the cell.

To add a stop point to a cell,

1. Choose *View – Parts Table*.

2. In the *Cell Bindings* section, right-click the cell to which you want to add a stop point.
3. From the pop-up menu, select *Add Stop Point*.

The 🛑 icon (stop icon) appears in the *Info* column of the cell row. All the instantiations of the cell will now be considered leaf nodes in the hierarchy; none of them will be expanded when the hierarchy is traversed.

To see a list of the instantiations to which the stop point applies,

1. In the *Cell Bindings* section, right-click the cell.
2. From the pop-up menu, select *Explain*.

   The Explain dialog box appears. The Instantiations section lists the instantiations of the cell. The stop point applies to all these instantiations.

Defining Bind-to-Open on a Per Cell Basis

You can specify that a single instantiation of a cell is to be skipped by setting a bind-to-open attribute on it.

Note: Only non-text instances, such as schematic instances can be skipped using bind-to-open. Bind-to-open should not be set on hierarchical text instances.

You can set the attribute on a cell in two ways:

- Bind-to-open on a cell
- Bind-to-open on a cell in a specific library

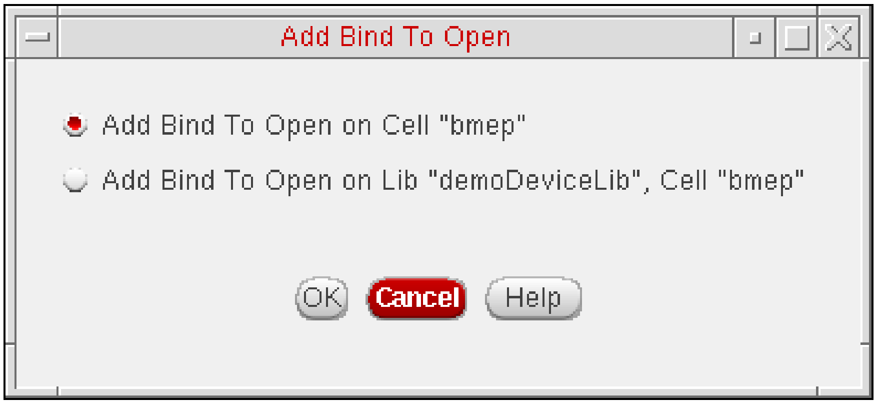
The Hierarchy Editor displays the string `**UNBOUND**` to indicate that a cell has a bind-to-open attribute. `**UNBOUND**` is not an error, unlike `**NONE*` . `**UNBOUND**` indicates that the cell is deliberately unbound while `**NONE**` indicates that the binding for the cell could not be determined from the binding rules.

To set a bind-to-open attribute on a cell,

1. Choose *View – Parts Table*.
2. In the *Cell Bindings* section, right-click the cell to which you want to add a bind-to-open attribute.
3. From the pop-up menu, select *Add Bind To Open (Skip Cell)*.

Note: In case multiple rows are selected, the pop-up menu will display the *Add Bind To Open (Skip Multiple Cells)* option.

The Add Bind To Open dialog box appears.

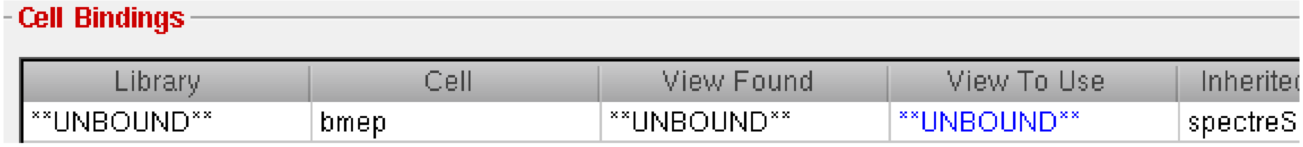


● Add Bind To Open on Cell "bmep"

○ Add Bind To Open on Lib "demoDeviceLib", Cell "bmep"

4. In the Add Bind To Open dialog box, do one of the following:
   - ❑ If you want the cell to be unbound, regardless of which library it comes from, select *Add Bind To Open on Cell "cellname"*.
   - ❑ If you want the cell to be unbound when it is obtained from a specific library, select *Add Bind To Open on Lib "libname"*, *Cell "cellname"*.
5. Click *OK*.

   The cell is now unbound. The *Library*, *View Found* and *View to Use* columns in the Cell Bindings section display `**UNBOUND**`. If you selected *Add Bind to Open on Library "libraryname", Cell "cellname"*, only the *View Found* and *View to Use* columns display `**UNBOUND**`. For example:



Cell Bindings

Library	Cell	View Found	View To Use	Inheritec
UNBOUND	bmep	**UNBOUND**	**UNBOUND**	spectreS

Also, the instances contained in the cell are no longer displayed in the Instance Bindings section because the cell is now unbound.

To see a list of the instantiations to which the bind-to-open attribute applies,

1. In the *Cell Bindings* section, right-click the cell.
2. From the pop-up menu, select *Explain*.

   The Explain dialog box appears. The Instantiations section lists the instantiations of the cell. The bind-to-open attribute applies to all these instantiations.

Defining Rules at the Instance Level

You can define rules at the instance level that override global rules and cell rules. Instance rules apply to only one instantiation of a cell, unlike cell-level rules that apply to all the instantiations of a cell. Note, however, that if the cell that contains the instance is used in multiple places in the design, the binding applies to the instance in all those locations.

Some instance-level rules can also apply to the hierarchy below the instance.

You specify instance-level rules, also referred to as instance bindings, in either of the following places:

- The *Instance Bindings* section in the Table View.
- The Tree View.

  In the tree view, any commands you enter (bindings, properties, stop points) are applied to the selected instance by default. If you want to apply the commands to an occurrence, select the *Occurrence Mode* button on the toolbar.

You can specify the following rules at the instance level:

- Library Binding
- View Binding
- Stop Point
- Bind-to-Open

Instance-level rules can be overridden at the occurrence level.

Changing View Bindings on a Per-Instance Basis

You can specify view bindings for a single instantiation of a cell. You can make the view binding apply to only the instance you specify (effective no lower in the hierarchy) or you can change it in a way that it is also inherited by the components below the instance in the hierarchy.

To make the binding apply only to the instance, you specify the view in the *View to Use* column. To make the binding applicable to components below the instance, you specify a view or view list in the *Inherited View List* column.

To display all of the instances for a particular cell,

1. In the *Cell Bindings* table, click the name of a cell.

2. From the toolbar, click the *Instance Table* button.

   The Hierarchy Editor displays the Instance Bindings table.

To specify an instance binding that applies only to the instance,

1. Display either the tree view or the table view of the Hierarchy Editor.
2. If you are displaying the tree view, verify that the *Occurrence Mode* button is deselected.

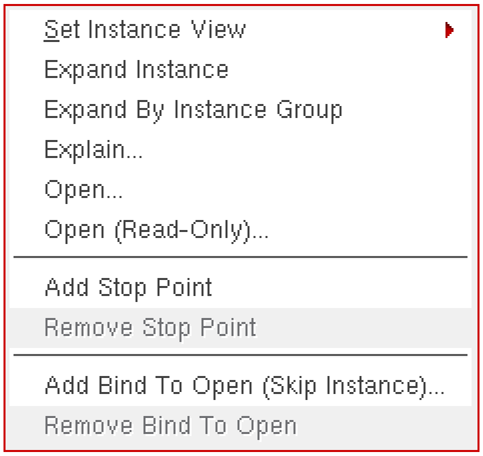
   The tree view now displays *Target: Instance*.
3. Do one of the following:

   a. Click in the *View to Use* column of the instance you want to change and type the new view name.

   – or –

   a. Right-click the instance you want to change.

   The following pop-up menu appears.



   b. Select *Set Instance View* and, from the list of available views, select the view you want.

   Note: The *Source File* and *Reference Verilog* options let you specify a text file. See "Using Text Files in Your Configuration" on page 180 for more information.

   The new view name appears in the *View to Use* column in the color used to display user bindings. This binding is not inheritable.

4. To see the results of your changes, click the *Update* icon in the toolbar.

If you selected the *Automatic Update* option in the Options form, this step is not required because your configuration is automatically updated.

5. To save the changes, choose *File – Save* or press `Control-s`.

To specify an instance binding that applies to the instance as well as to objects below it in the hierarchy,

1. Display either the tree view or the table view of the Hierarchy Editor.
2. If you are displaying the tree view, verify that the *Occurrence Mode* button is deselected.

   The tree view now displays *Target: Instance*.
3. Click in the *Inherited View List* column of the instance you want to change, type a new view or list of views, and press `Return`.

   You can use constants in the view list; see "Using Constants" on page 189 for more information. You can also use the asterisk character (`*`) as a wildcard in the view list; see "Using Wildcards in a View List" on page 193 for more information.
4. To see the results of your changes, click the *Update* icon in the toolbar.

   If you selected the *Automatic Update* option in the Options form, this step is not required because your configuration is automatically updated.
5. To save the changes, from the menu bar, click *File – Save* or press `Control-s`.

Changing Library Bindings on a Per-Instance Basis

You can define an instance-level library list that overrides the global and cell library lists for a single instantiation of a cell. The library list determines the library from which the instance is obtained. Libraries are listed in order of preference—if the instance is found in the first library in the list, it is used; if it is not found, the second library is searched, and so on.

The instance-level library list:

- Applies to only one instantiation of a cell
- Can apply to multiple objects—if the cell that contains the instance is used in multiple places in the design, the library binding applies to the instance in all those locations
- Is inherited by all components in the hierarchy below the instance

An instance level library binding can be overridden by an occurrence binding.

To create an instance-level library list,

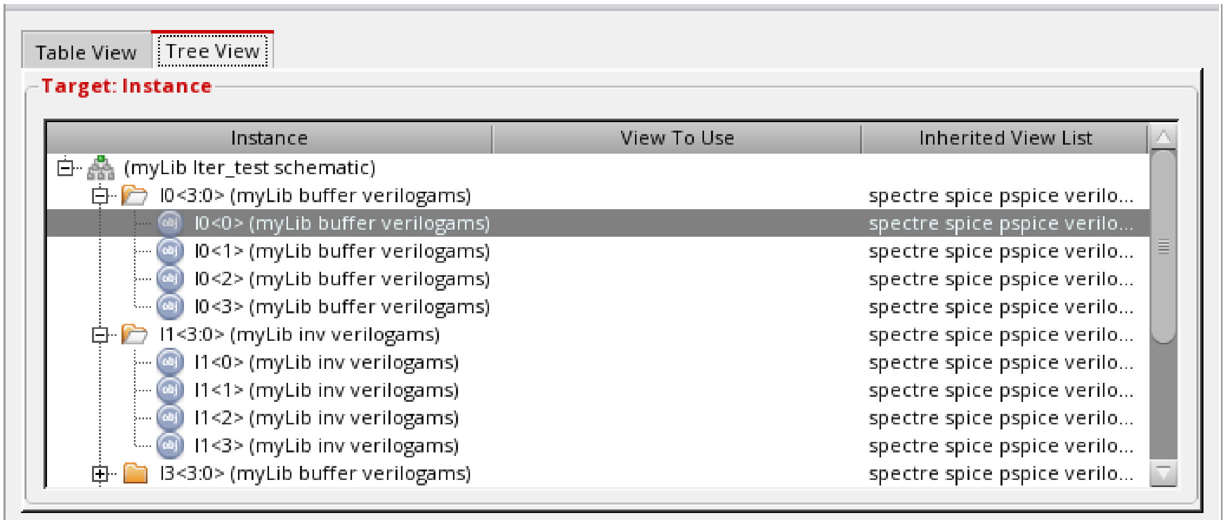
1. Choose *View – Parts Table*.
2. In the *Cell Bindings* table, select the cell that contains the instance you want to change.
3. In the *Instance Bindings* section, click in the *Inherited Lib List* column of the instance whose binding you want to change.

   The field becomes editable. If the field does not become editable, the library is fixed for your design and cannot be changed.
4. Edit the inherited library list. The library list that is currently displayed is the library list inherited from higher levels of the hierarchy. Type the libraries in order of preference.
5. Press `Return`.
6. To view the changes in the configuration, click the *Update* icon in the toolbar.

   If you selected the *Automatic Update* option in the Options form, this step is not required because your configuration is automatically updated.
7. To save the changes, choose *File – Save* or press `Control-s`.

To set view bindings separately for individual bits of an iterated instance,

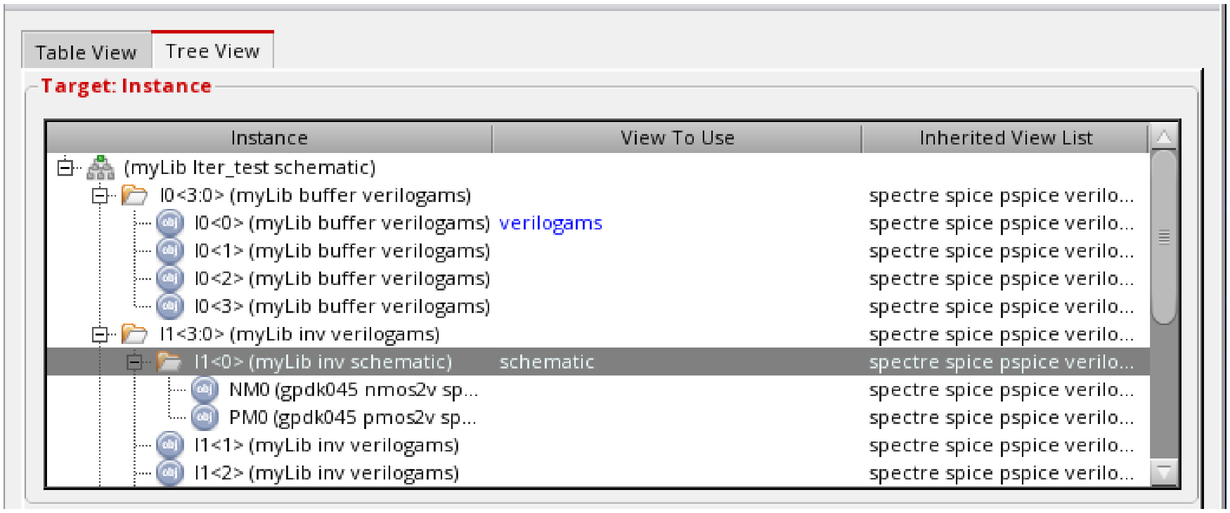
1. Open the view in Hierarchy Editor and switch to the Tree View.



Instance	View To Use	Inherited View List
(myLib Iter_test schematic)		
I0<3:0> (myLib buffer verilogams)		spectre spice pspice verilo...
I0<0> (myLib buffer verilogams)		spectre spice pspice verilo...
I0<1> (myLib buffer verilogams)		spectre spice pspice verilo...
I0<2> (myLib buffer verilogams)		spectre spice pspice verilo...
I0<3> (myLib buffer verilogams)		spectre spice pspice verilo...
I1<3:0> (myLib inv verilogams)		spectre spice pspice verilo...
I1<0> (myLib inv verilogams)		spectre spice pspice verilo...
I1<1> (myLib inv verilogams)		spectre spice pspice verilo...
I1<2> (myLib inv verilogams)		spectre spice pspice verilo...
I1<3> (myLib inv verilogams)		spectre spice pspice verilo...
I3<3:0> (myLib buffer verilogams)		spectre spice pspice verilo...

2. Expand an iterated instance by clicking the + icon preceding it.
3. Right-click each bit for which you want to change the view bindings and choose *Set Instance View – <view name>*.

The *View To Use* column in the Tree View is updated with the new binding information.



Instance	View To Use	Inherited View List
(myLib Iter_test schematic)		
I0<3:0> (myLib buffer verilogams)		spectre spice pspice verilo...
I0<0> (myLib buffer verilogams)	verilogams	spectre spice pspice verilo...
I0<1> (myLib buffer verilogams)		spectre spice pspice verilo...
I0<2> (myLib buffer verilogams)		spectre spice pspice verilo...
I0<3> (myLib buffer verilogams)		spectre spice pspice verilo...
I1<3:0> (myLib inv verilogams)		spectre spice pspice verilo...
I1<0> (myLib inv schematic)	schematic	spectre spice pspice verilo...
NM0 (gpdk045 nmos2v sp...		spectre spice pspice verilo...
PM0 (gpdk045 pmos2v sp...		spectre spice pspice verilo...
I1<1> (myLib inv verilogams)		spectre spice pspice verilo...
I1<2> (myLib inv verilogams)		spectre spice pspice verilo...

In the preceding screenshot, iterated instance bit I1<0> is bound to the schematic view whereas the rest of the bits of iterated instance I1<3:0> are bound to verilogams.

4. Click *Recompute the hierarchy* on the toolbar.

You can similarly use the Table View to set iterated instance bit bindings.

Note: Iterated instance bit bindings are currently only supported by the AMS UNL netlister.

Defining Stop Points on a Per-Instance Basis

You can specify a stop point on a single instantiation of a cell that prevents the instance from being expanded when the hierarchy is expanded.

A stop point on an instance can apply to multiple objects—if the cell that contains the instance is used in multiple places in the design, the stop point applies to the instance in all those places. For example, if you put a stop point on instance `I2` of cell `OpAmp` and cell `OpAmp` is used in three places in the design, the stop point applies to instance `I2` in all three instantiations of `OpAmp`. (If you want to specify a stop point on a single object, see "Defining Occurrence Stop Points" on page 177 for information.)

You can add a stop point to an instance from either the tree view or the table view of the Hierarchy Editor.

To add a stop point to an instance from the tree view,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. If the *Occurrence Mode* button on the toolbar is selected, deselect it.

   The tree view now displays *Target: Instance*.
3. Right-click the instance on which you want to add a stop point.
4. From the pop-up menu, select *Add Stop Point*.

   The instance icon changes to a stop point icon. The instance can no longer be expanded.

   Note: If the instance has already been identified as an occurrence and the tree view displays *Target: Occurrence* when the instance is selected, the stop point is added to the occurrence. For more information about occurrence stop points, see "Defining Occurrence Stop Points" on page 177.

To add a stop point to an instance from the table view,

1. Choose *View – Parts Table* and *View – Instance Table* to display the table view of the Hierarchy Editor, if it is not already displayed.
2. In the *Cell Bindings* section, select the cell that contains the instance to which you want to add a stop point.

   The *Instance Bindings* section displays the instances contained in the cell.
3. In the *Instance Bindings* section, right-click the instance to which you want to add a stop point.
4. From the pop-up menu, select *Add Stop Point*.

   The stop point icon appears in the *Info* column of the instance row. The instance can no longer be expanded.

If the cellview that contains the instance is used in multiple places in the design, the stop point will apply to the instance in all those places.

To see which objects the stop point applies to,

1. In the *Cell Bindings* section, right-click the cell that contains the instance to which you added the stop point.
2. From the pop-up menu, select *Explain*.

   The Explain dialog box appears. The *Instantiations* section lists the instantiations of the cell. The stop point will apply to the instance in all these instantiations of the cell.

Note: You cannot add an instance stop point to an object that has already been defined as an occurrence. When you add a stop point to such an object, it is added to the occurrence, not the instance, and applies only to one object at a specific path. For information about occurrence stop points, see "Defining Occurrence Stop Points" on page 177.

Defining Bind-to-Open on a Per-Instance Basis

You can specify that a single instantiation of a cell is to be skipped by setting a bind-to-open attribute on it.

Note: Only non-text instances, such as schematic instances can be skipped using bind-to-open. Bind-to-open should not be set on hierarchical text instances.

A bind-to-open attribute on an instance can apply to multiple objects—if the cell that contains the instance is used in multiple places in the design, the bind-to-open applies to the instance in all those places. For example, if you specify that instance `I2` in cell `OpAmp` is unbound and cell `OpAmp` is used in three places in the design, `I2` will be unbound in all three instantiations of `OpAmp`. (If you want to set a bind-to-open attribute on a single object, see "Defining Occurrence-Level Bind-to-Open" on page 178 for information.)

The Hierarchy Editor displays the string `**UNBOUND**` to indicate that an instance has a bind-to-open attribute. `**UNBOUND**` is not an error, unlike `**NONE**`. `**UNBOUND**` indicates that the instance is deliberately unbound while `**NONE**` indicates that the binding for the instance could not be determined from the binding rules.

You can add a bind-to-open attribute to an instance from either the tree view or the table view of the Hierarchy Editor.

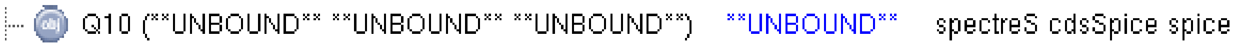
To add a bind-to-open attribute to an instance from the tree view,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. If the *Occurrence Mode* button on the toolbar is selected, deselect it.

   The tree view now displays *Target: Instance*.
3. Right-click the instance on which you want to add a bind-to-open.
4. From the pop-up menu, select *Add Bind To Open (Skip Instance)*.

Note: In case multiple rows are selected, the pop-up menu will display the *Add Bind To Open (Skip Multiple)* option.

The instance is now unbound. The library, cell, and view name of the instance are replaced by `**UNBOUND**`. For example:



```
Q10 (**UNBOUND** **UNBOUND** **UNBOUND**)   **UNBOUND**   spectreS cdsSpice spice
```

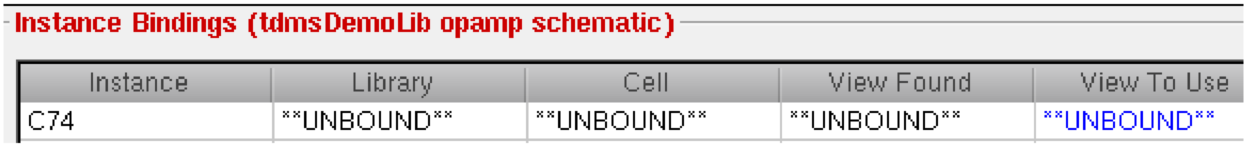
Note: If the instance has already been identified as an occurrence and the tree view displays *Target: Occurrence* when the instance is selected, the bind-to-open attribute is applied to the occurrence. For more information about occurrence-level bind-to-open, see "Defining Occurrence-Level Bind-to-Open" on page 178.

To add a bind-to-open attribute to an instance from the table view,

1. Choose *View – Parts Table* and *View – Instance Table* to display the table view of the configuration, if it is not already displayed.
2. In the *Cell Bindings* section, select the cell that contains the instance to which you want to add a bind-to-open attribute.

   The *Instance Bindings* section displays the instances contained in the cell.
3. In the *Instance Bindings* section, right-click the instance to which you want to add a bind-to-open.
4. From the pop-up menu, select *Add Bind To Open (Skip Instance)*.

   The instance is now unbound. The *Library*, *Cell*, *View Found*, and *View to Use* columns display `**UNBOUND**`. For example:



Instance Bindings (tdmsDemoLib opamp schematic)

Instance	Library	Cell	View Found	View To Use
C74	**UNBOUND**	**UNBOUND**	**UNBOUND**	**UNBOUND**

If the cellview that contains the instance is used in multiple places in the design, the bind-to-open will apply to the instance in all those places.

To see which objects the bind-to-open applies to,

1. In the *Cell Bindings* section, right-click the cell that contains the instance to which you added the bind-to-open.
2. From the pop-up menu, select *Explain*.

   The Explain dialog box appears. The *Instantiations* section lists the instantiations of the cell. The bind-to-open will apply to the instance in all these instantiations of the cell. If any of the instances has an occurrence binding on it already, the bind-to-open will not apply to it.

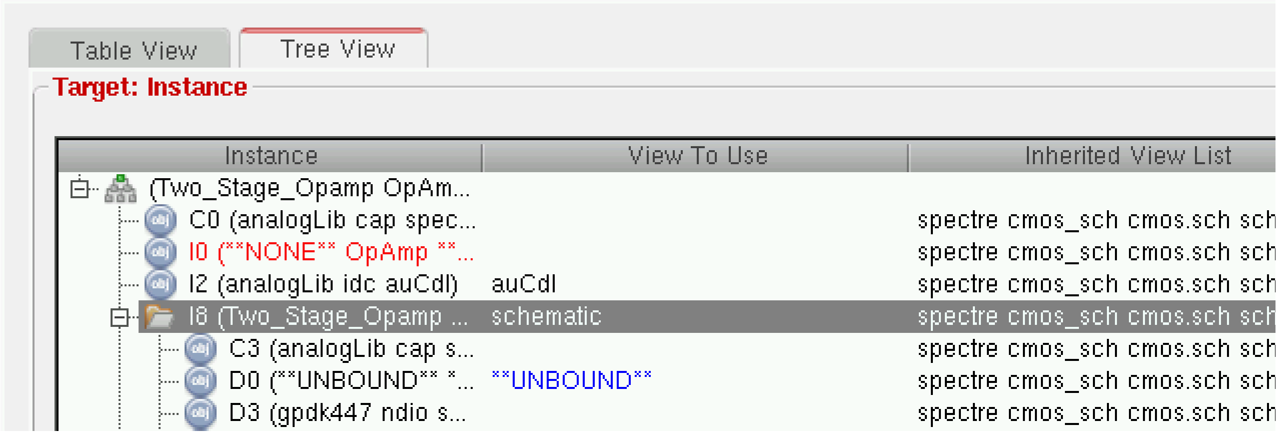
Changing Instance Bindings Inside a Block

A block is a group of instances used in simulators like VHDL. Blocks can contain other blocks and other instances. You can bind instances inside a block the same way you can bind instances in a cellview.

Blocks are better displayed in the tree structure of the Hierarchy Editor.

To change the instance bindings inside a block,

1. Select the *Tree View* tab to display the tree view.
2. Select the block containing the instance you want to change.
3. Expand the block to display the instances within it.



Instance	View To Use	Inherited View List
(Two_Stage_Opamp OpAm...		
C0 (analogLib cap spec...		spectre cmos_sch cmos.sch sch
I0 (**NONE** OpAmp **...		spectre cmos_sch cmos.sch sch
I2 (analogLib idc auCdl)	auCdl	spectre cmos_sch cmos.sch sch
I8 (Two_Stage_Opamp ...	schematic	spectre cmos_sch cmos.sch sch
C3 (analogLib cap s...		spectre cmos_sch cmos.sch sch
D0 (**UNBOUND** *...	**UNBOUND**	spectre cmos_sch cmos.sch sch
D3 (gpdk447 ndio s...		spectre cmos_sch cmos.sch sch

4. Right-click the instance whose bindings you want to change.

   The pop-up menu appears.

5. From the pop-up menu, select *Set Instance View – newView*.

   The new view name appears in the *View to Use* column in the color used to display user bindings.

6. To see the results of your changes, click the *Update* icon in the toolbar.

   If you selected the *Auto Update* option in the Options form, this step is not required because your configuration is automatically updated.

7. To save the changes, from the menu bar, click *File – Save* or press `Control-s`.

Defining Rules at the Occurrence Level

About Occurrences

An occurrence is an instance that is uniquely identified by its full path from the top-level design to the instance. Occurrences provide the ability to uniquely identify an object in the design and assign attributes to that object.

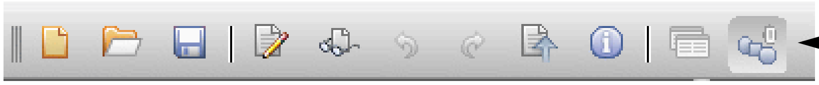
Occurrence binding is particularly useful for mixed-signal simulation.

In the Hierarchy Editor, you can assign the following attributes to an occurrence:

- Occurrence binding
- Occurrence stop point
- Occurrence-level bind-to-open

Setting any of the above attributes identifies the object as an occurrence.

The Hierarchy Editor provides an occurrence editing mode in the tree view. You can turn on the mode by selecting the following button in the toolbar:



Click to set occurrence editing mode

The tree view displays *Target: Occurrence* when you select any object in the tree to indicate that any change you make will be applied to the occurrence.

In the Hierarchy Editor, occurrences are represented by the following icons:

The object has an occurrence binding

The tree node contains an occurrence

The object has an occurrence stop point

The object has an occurrence binding and an occurrence stop point

To see a complete list of icons and color conventions used in the Virtuoso Hierarchy Editor,

➤ Choose *Help – Legend*.

The Legend dialog box appears. This dialog box describes all the icons and color conventions.

Note: Not all Cadence tools support occurrences. Before you use the occurrence feature of the Hierarchy Editor, refer to the documentation for the applications you are using to check if those applications support occurrences.

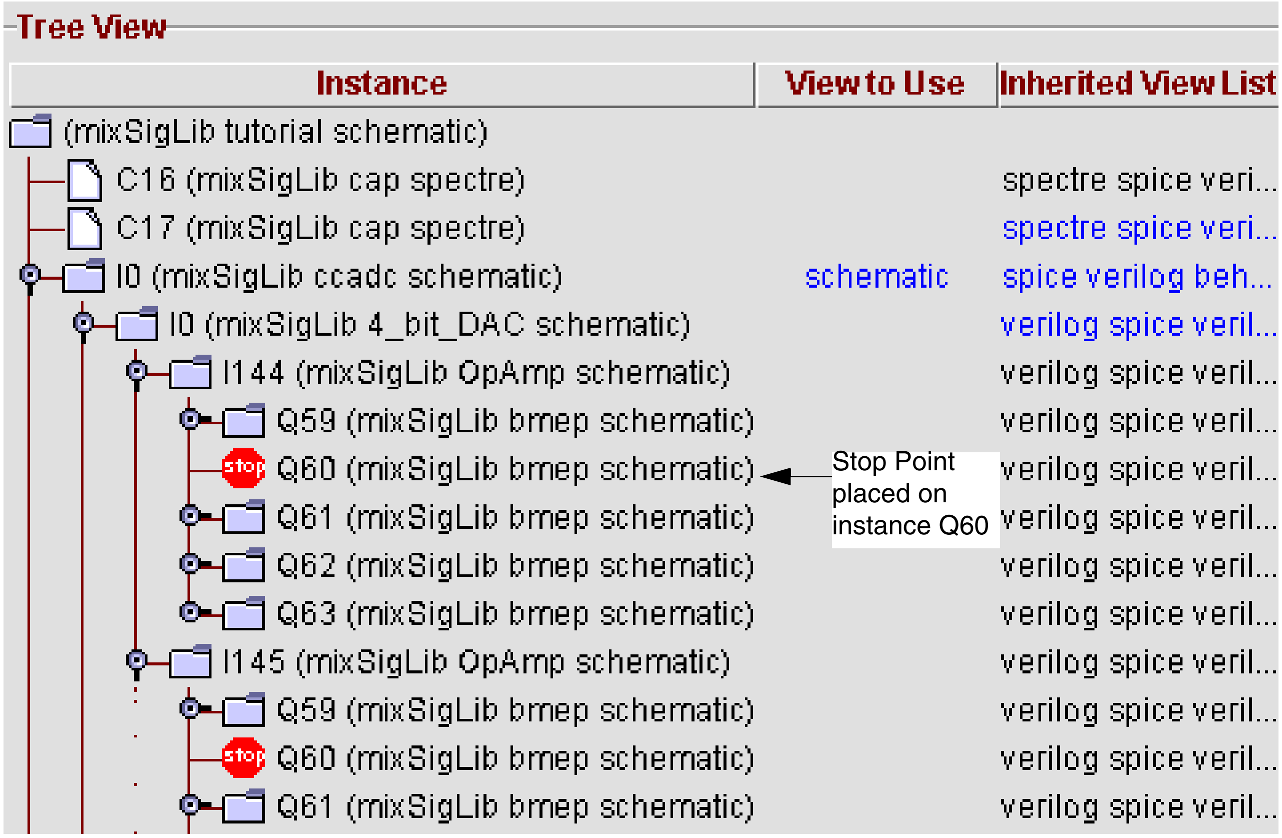
How Occurrences Are Different from Instances

Occurrences are unique, while instances are not. When you put an attribute (such as a stop point) on an instance, it can apply to more than one object because instances cannot be identified uniquely. An instance is identified by the cell it is contained in. If the cell is used in multiple places in the design, the instance is in multiple places in the design.

An occurrence, on the other hand, is identified uniquely by its full path from the top-level design. When you put an attribute (such as a stop point) on an occurrence, it only applies to that single object.

The following example illustrates the difference between instances and occurrences. Both Figure 1 and Figure 2 display the design `mixSigLib.tutorial:schematic`. In this design, the cell `OpAmp` is used twice—its instantiations are `mixSigLib.tutorial:schematic.I0.I0.I144` and `mixSigLib.tutorial:schematic.I0.I0.I145`. The cell `OpAmp` contains several instances, including `Q60`.

In Figure 1, an instance stop point is specified on `Q60` in `I144`. Notice that `Q60` in `I145` (the other instantiation of cell `OpAmp`) automatically gets the stop point, too.



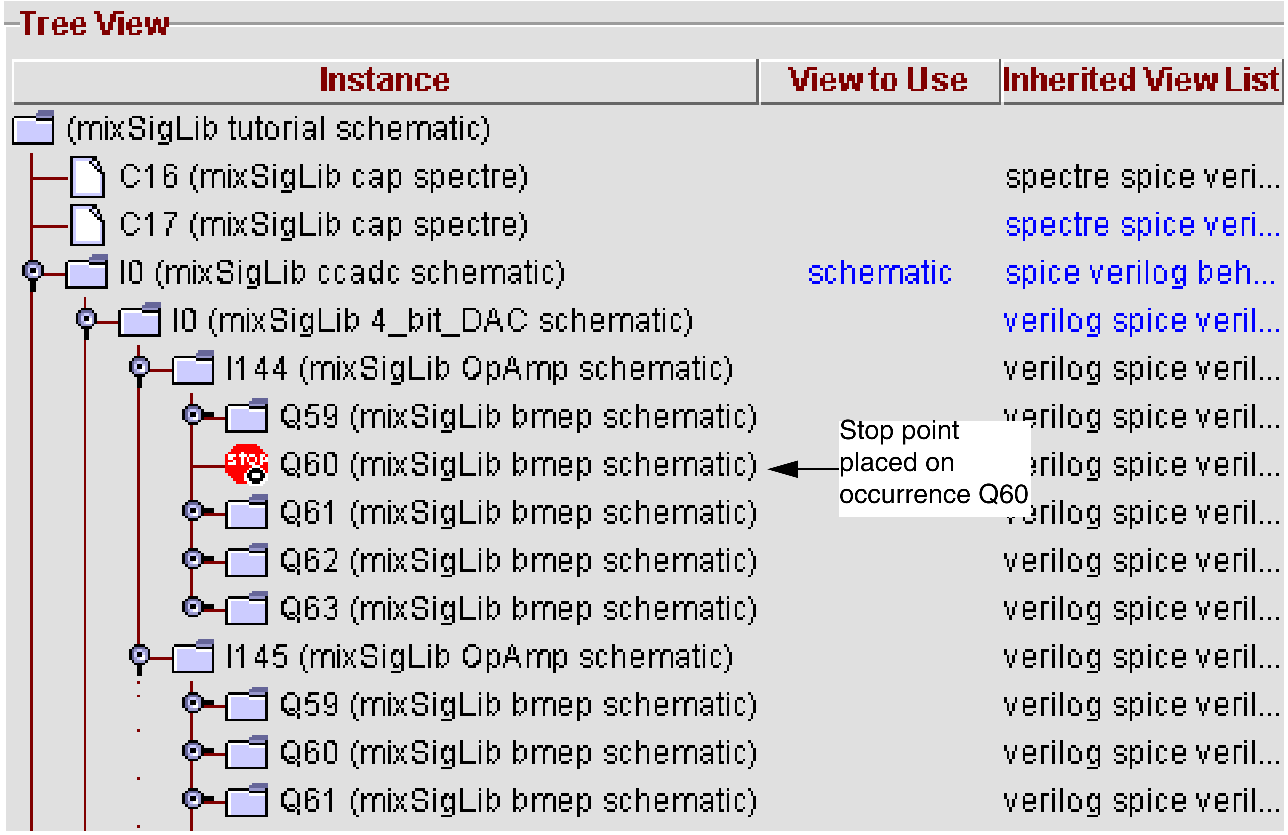
Tree View

Instance	View to Use	Inherited View List
(mixSigLib tutorial schematic)		
C16 (mixSigLib cap spectre)		spectre spice veri...
C17 (mixSigLib cap spectre)		spectre spice veri...
I0 (mixSigLib ccadc schematic)	schematic	spice verilog beh...
I0 (mixSigLib 4_bit_DAC schematic)		verilog spice veril...
I144 (mixSigLib OpAmp schematic)		verilog spice veril...
Q59 (mixSigLib bmep schematic)		verilog spice veril...
stop Q60 (mixSigLib bmep schematic)		verilog spice veril...
Q61 (mixSigLib bmep schematic)		verilog spice veril...
Q62 (mixSigLib bmep schematic)		verilog spice veril...
Q63 (mixSigLib bmep schematic)		verilog spice veril...
I145 (mixSigLib OpAmp schematic)		verilog spice veril...
Q59 (mixSigLib bmep schematic)		verilog spice veril...
stop Q60 (mixSigLib bmep schematic)		verilog spice veril...
Q61 (mixSigLib bmep schematic)		verilog spice veril...

Stop Point placed on instance Q60

Figure 1: Example of an Instance-Level Stop Point

In Figure 2, an occurrence stop point is specified on `Q60` in `I144`. Notice that `Q60` in `I145` (the other instantiation of cell `OpAmp`) is not affected by the stop point. This is because the stop point was put on the occurrence and not on the instance.



Tree View

Instance	View to Use	Inherited View List
(mixSigLib tutorial schematic)		
C16 (mixSigLib cap spectre)		spectre spice veri...
C17 (mixSigLib cap spectre)		spectre spice veri...
I0 (mixSigLib ccadc schematic)	schematic	spice verilog beh...
I0 (mixSigLib 4_bit_DAC schematic)		verilog spice veril...
I144 (mixSigLib OpAmp schematic)		verilog spice veril...
Q59 (mixSigLib bmep schematic)		verilog spice veril...
Q60 (mixSigLib bmep schematic)		erilog spice veril...
Q61 (mixSigLib bmep schematic)		erilog spice veril...
Q62 (mixSigLib bmep schematic)		verilog spice veril...
Q63 (mixSigLib bmep schematic)		verilog spice veril...
I145 (mixSigLib OpAmp schematic)		verilog spice veril...
Q59 (mixSigLib bmep schematic)		verilog spice veril...
Q60 (mixSigLib bmep schematic)		verilog spice veril...
Q61 (mixSigLib bmep schematic)		verilog spice veril...

Stop point placed on occurrence Q60

Figure 2: Example of an Occurrence Stop Point

Defining Occurrence Bindings

You create an occurrence binding by specifying a library, cell, and view—or a text file—for a single object at a specific path in the design. Unlike cell or instance binding, occurrence binding applies only to one object.

When you do an occurrence binding, none of the other binding rules—such as inherited library lists or view lists—apply to that occurrence.

You can create an occurrence binding only in the tree view of the Hierarchy Editor, in the occurrence editing mode; you cannot create it in the table view.

Note: If the library or cell is fixed by some other rules in the design data, you will not be able to change that binding and will only be able to set the view. For example, design data created by the Virtuoso schematic editor typically does not allow you to change the library and cell.

Creating an Occurrence Binding

You can bind an occurrence to

- A library, cell, and view

  Note: If the library and cell are fixed by some other rules in the design data, you can only set the view.

- A source file
- A Verilog file

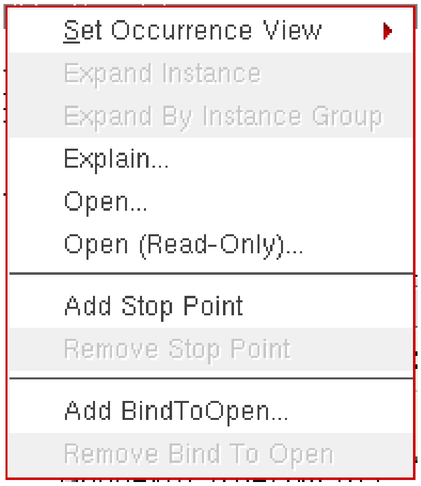
This section describes how to bind an occurrence to a library/cell/view. For information on how to bind an occurrence to a source file or a Verilog file, see "Using Text Files in Your Configuration" on page 180.

To specify a library/cell/view binding for an occurrence,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the Set bindings...on Occurrences button in the toolbar:

   The tree view displays *Target: Occurrence* as the section header.
3. Right-click the object to which you want to add an occurrence binding.

The following pop-up menu appears:



4. From the pop-up menu, Select *Set Occurrence View* and select a view from the list of available views.

   The Hierarchy Editor sets the view binding to the view you specify and the library and cell binding to the current library and cell.

   Note: You can also set the view by typing it in the *View to Use* column of the occurrence.

The object is now identified as an occurrence and a library/cell/view binding is set on it. The leaf node or tree node icon that was displayed for the object is replaced by the icon (occurrence icon). If the object already had an occurrence stop point, the icon is replaced by the icon (occurrence binding and stop point icon).

Editing an Occurrence Binding

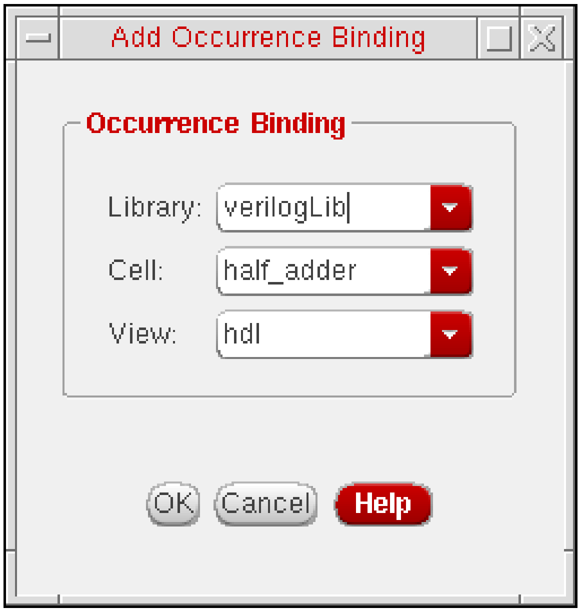
To edit an occurrence binding,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the following button in the toolbar:

   The tree view displays *Target: Occurrence*.
3. Select the occurrence whose binding you want to change.
4. If you want to change only the view, do one of the following:
   - Right-click the occurrence, select *Set Occurrence View* on the pop-up menu, and then select the new view from the list of available views.
   - Type the new view in the *View to Use* column of the selected occurrence.

   Skip the remaining steps in this section.
5. If you want to change the library or cell,
   1. Right-click the occurrence.
   2. From the pop-up menu that appears, select *Set Occurrence View – Library/Cell/View*.

      The Add Occurrence Binding dialog box appears.



Add Occurrence Binding

Occurrence Binding

Library: verilogLib

Cell: half_adder

View: hdl

Note: If the library and cell are fixed for the design data, then the *Set Occurrence View – Library/Cell/View* option is not available. You can only specify a view binding. For example, Virtuoso design data typically does not allow you to change the library and cell.

c. Edit the library, cell, and view to which the occurrence is bound.

d. Click *OK*.

The dialog box closes.

To edit an occurrence binding to a source file,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the *Set bindings...on occurrence* button in the toolbar:

   The tree view displays *Target: Occurrence*.
3. Right-click the occurrence whose binding you want to change.
4. From the pop-up menu, select *Set Occurrence View – Source File*.

   The Enter the sourcefile Location form appears. For information on specifying a source file, see "Using Source Files in Your Configuration" on page 180.

To edit an occurrence binding to a Verilog file,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the *Set bindings...on occurrence* button in the toolbar:

   The tree view displays *Target: Occurrence*.
3. Right-click the occurrence whose binding you want to change.
4. From the pop-up menu, select *Set Occurrence View – Reference Verilog*.

   The Reference Verilog Modules form appears. For information on specifying a Verilog file, see "Referencing a Verilog File" on page 186.

Removing an Occurrence Binding

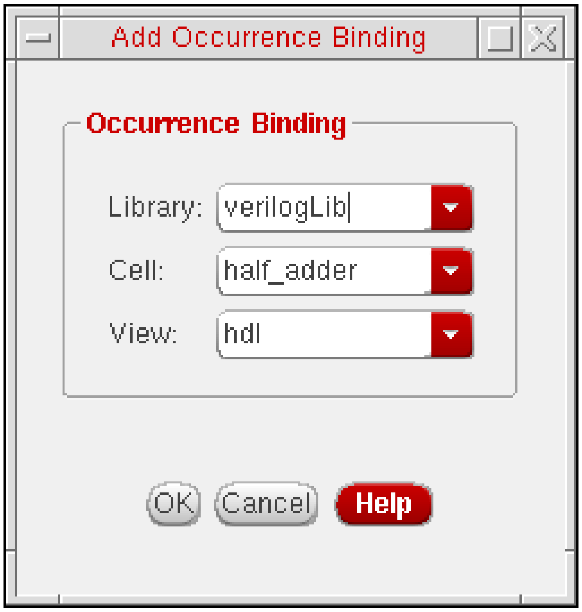
To remove an occurrence binding,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the *Set bindings…on occurrence* button in the toolbar:

   The tree view displays *Target: Occurrence*.
3. Right-click the occurrence whose binding you want to remove.
4. From the pop-up menu, select *Set Occurrence View – <none>*.

The occurrence binding is removed from the object. The icon (occurrence icon) is replaced by the leaf or folder icon. The object is no longer uniquely identified, unless it also has an occurrence stop on it.

About the Add Occurrence Binding Form



Library: verilogLib

Cell: half_adder

View: hdl

Occurrence Binding

Library lets you type or select the library to which you want to bind the occurrence.

Cell lets you type or select the cell to which you want to bind the occurrence.

View lets you type or select the view to which you want to bind the occurrence.

OK applies your changes and closes the form.

Cancel cancels your changes and closes the form.

Help opens this manual.

Defining Occurrence Stop Points

An occurrence stop point is a stop point on a specific path and applies only to one object in the design.

If an object has already been defined as an occurrence (an object is defined as an occurrence if it has one of the following occurrence attributes: occurrence binding, occurrence stop point, or occurrence bind-to-open), when you add a stop point, you are automatically adding it to the occurrence, not to the instance.

You can only add an occurrence stop point in the tree view of the Hierarchy Editor; you cannot add it in the table view.

Adding an Occurrence Stop Point

To add an occurrence stop point,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the *Set bindings...on occurrence* button in the toolbar:

   The tree view displays *Target: Occurrence*.
3. Right-click the object on which you want to add a stop point.
4. From the pop-up menu, select *Add Stop Point*.

The icon (occurrence stop point icon) appears next to the occurrence. The occurrence cannot be expanded until the stop point is removed.

Removing an Occurrence Stop Point

To remove an occurrence stop point,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Right-click the occurrence.
3. From the pop-up menu, select *Remove Stop Point*.

The occurrence stop point is removed. The icon is replaced by the leaf or folder icon. The object is no longer uniquely identified, unless it also has an occurrence binding.

Defining Occurrence-Level Bind-to-Open

You can specify that a single instantiation of a cell is to be skipped by setting a bind-to-open attribute on it.

Note: Only non-text instances, such as schematic instances can be skipped using bind-to-open. Bind-to-open should not be set on hierarchical text instances.

The Hierarchy Editor displays the string `**UNBOUND**` to indicate that an occurrence has a bind-to-open attribute. `**UNBOUND**` is not an error, unlike `**NONE**`. `**UNBOUND**` indicates that the occurrence is deliberately unbound while `**NONE**` indicates that the binding for the occurrence could not be determined from the binding rules.

You can only add an occurrence-level bind-to-open attribute in the tree view of the Hierarchy Editor; you cannot add it in the table view.

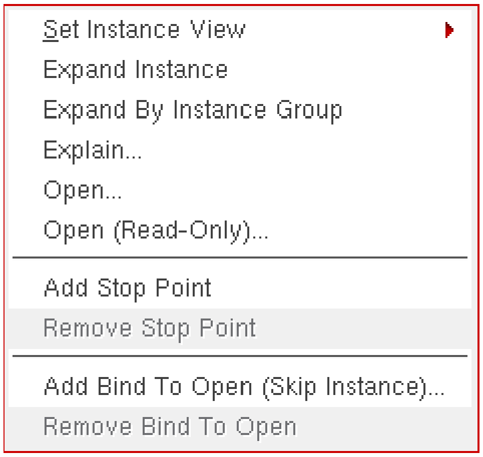
Setting Bind-to-Open on an Occurrence

To add a bind-to-open attribute to an occurrence,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Turn on the occurrence editing mode by clicking the following *Set bindings...on occurrence* in the toolbar:

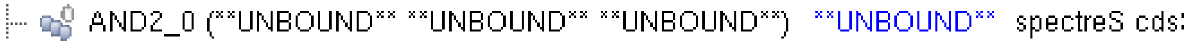
   The tree view displays *Target: Occurrence*.
3. Right-click the object on which you want to add an occurrence-level bind-to-open.

   The following pop-up menu appears:



4. From the pop-up menu, select *Add Bind To Open (Skip Instance)*.

The bind-to-open attribute is set on the occurrence. The icon (occurrence icon) appears next to the occurrence and the library, cell, and view names are replaced by `**UNBOUND**`. For example:



AND2_0 (**UNBOUND** **UNBOUND** **UNBOUND**) **UNBOUND** spectreS cds:

Removing Occurrence-Level Bind-to-Open

To remove an occurrence-level bind-to-open attribute,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. Right-click the occurrence from which you want to remove the bind-to-open attribute.
3. From the pop-up menu, select *Remove Bind To Open*.

The occurrence is no longer unbound. The library, cell, and view name are determined based on the other binding rules and displayed.

Using Text Files in Your Configuration

In your design configuration, you can also use text files that contain source descriptions of design elements. Instead of binding a cell, instance, or occurrence to a view, you can specify that a text file be used. These file bindings are similar to view bindings.

You can use the following types of text files in a configuration:

- Source files for HSPICE or SPICE design blocks
- Verilog files

Using Source Files in Your Configuration

You can use source files for HSPICE or SPICE design blocks in your design configuration. You use a source file by binding a cell, instance, or occurrence to the source file. This file binding is similar to a view binding.

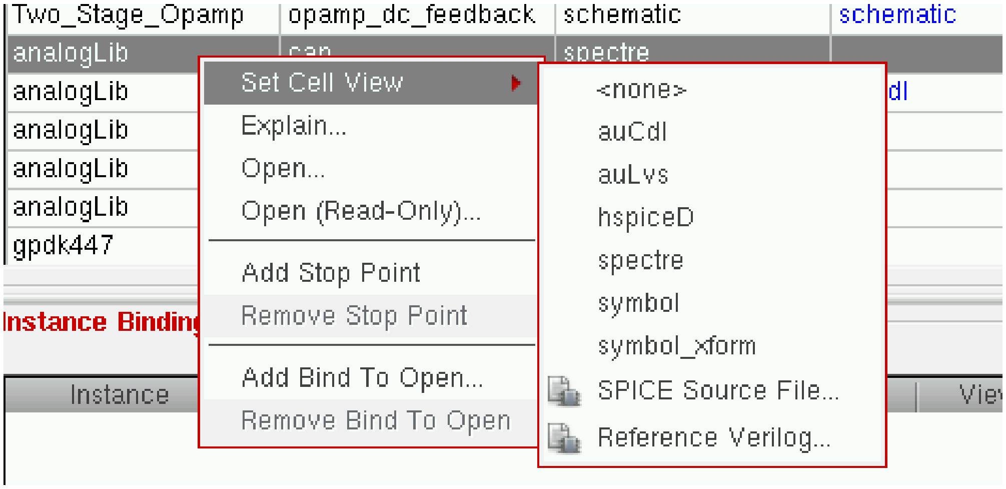
Binding to a source file puts the the `sourcefile` property on the cell, instance, or occurrence. The value of the property is the path to the source file and it is stored in the `prop.cfg` file in the configuration view. The `sourcefile` property is used by other applications in the flow.

To bind a cell, instance, or occurrence to a source file,

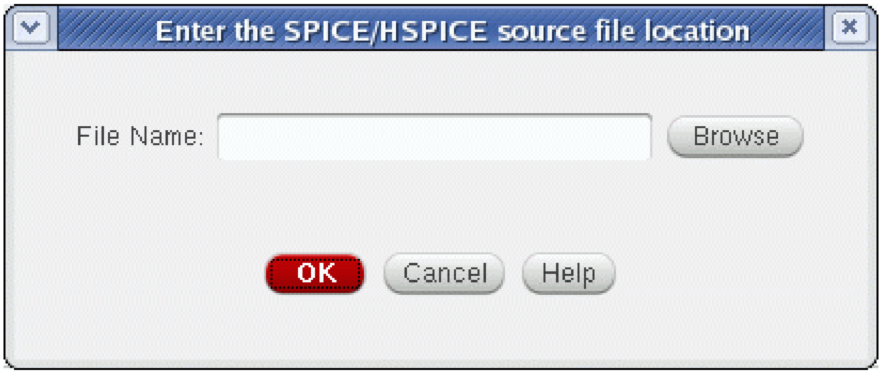
1. If you are binding a cell, display the table view of the Hierarchy Editor; if you are binding an instance, display either the table view or the tree view; if you are binding an occurrence, display the tree view and turn on the occurrence editing mode.
2. Right-click the cell, instance, or occurrence.

   Note: Even if you are not in occurrence editing mode, when you edit an object in the tree view that has been identified as an occurrence (by having an occurrence binding, occurrence stop point, or occurrence bind-to-open on it), you are editing the occurrence, not the instance. The tree view displays *Target: Occurrence* when you select such an object.

3. From the pop-up menu, select *Set Cell View / Set Instance View / Set Occurrence View – SPICE Source File*. For example:



The Enter the SPICE/HSPICE source file Location form appears.



4. In the form, do one of the following:

   ❑ Type the path to the source file in the *File Name* field. You can use environment variables in the path.

   ❑ Click *Browse* and use the Select the source file value form to select the file.

5. Click *OK*.

The cell, instance, or occurrence is bound to the source file. The *View to Use* column displays the path to the file. The 📄 icon next to the path indicates that it is a file binding.

To remove a source file binding,

1. If you are editing a cell, display the table view of the Hierarchy Editor; if you are editing an instance, display either the table view or the tree view; if you are editing an occurrence, display the tree view and turn on the occurrence editing mode.
2. Right-click the cell, instance, or occurrence.
3. From the pop-up menu, select *Set Cell View / Set Instance View / Set Occurrence View – <none>*.

Using Verilog Files in Your Configuration

You can use Verilog views in a design configuration in the same way that you use other views. Verilog views contain Verilog text files such as `verilog.v`, `verilog.vams`, or `verilog.va`, which describe Verilog modules.

However, if you have Verilog text files that are not in a Cadence design library, that is, they are not in the Cadence library/cell/view structure, you can still use them in your configuration in the following ways:

- By using the Hierarchy Editor's *Populate Library* command to bring Verilog modules from these text files into a Cadence design library.

  The Hierarchy Editor creates cellviews for all the modules in the library you specify. Use this option when you want the cellviews to be created in a master library or an explicit temporary directory.

- By referencing the Verilog text file.

  The file is later compiled, by other tools in the flow, into the implicit temporary directory. Use this option when you cannot create cellviews in the master library or explicit temporary directory.

Populating a Library with Verilog Modules

You can bring Verilog modules into your Cadence design library so that you can use them in your design configurations. The Hierarchy Editor creates a cellview in the library for each module, with corresponding `pc.db` and `master.tag` files, as well as a link to the original source file.

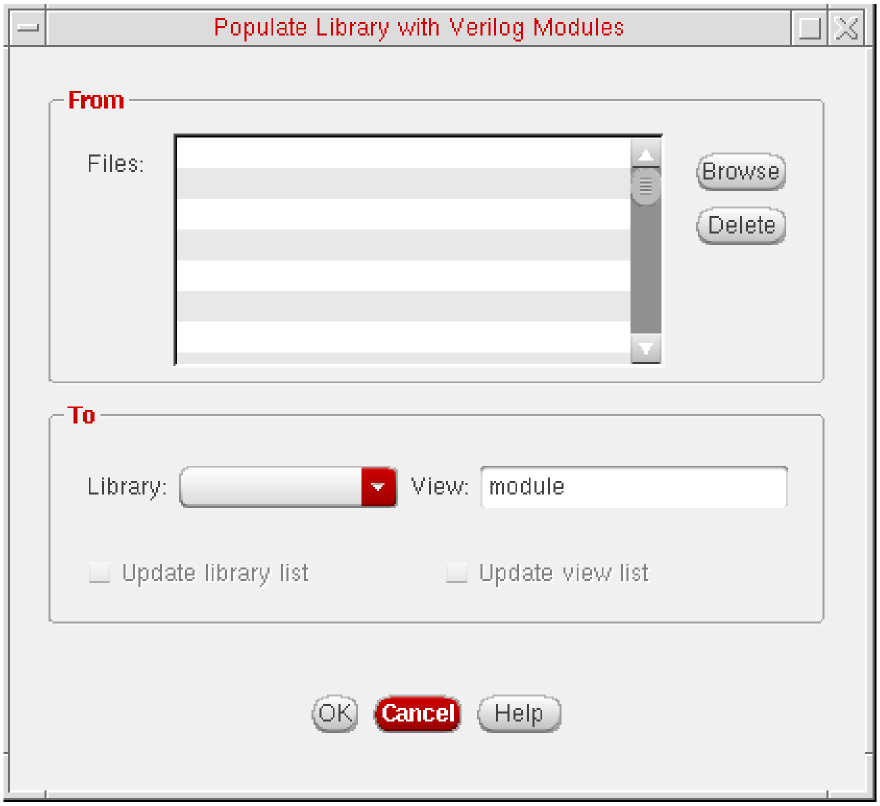
You can get modules from multiple Verilog source files at the same time. Each Verilog file can contain a single module or several modules that are independent of each other or hierarchically related.

You do not have to open a configuration to populate a library with Verilog modules. If you do open a configuration, you have the choice of updating the global view list and library list of the configuration with the Verilog views or the library you specify.

To populate a library with Verilog modules,

1. Choose *File – Populate Library – Verilog*.

The Populate Library with Verilog Modules form appears.



Populate Library with Verilog Modules

From

Files:

To

Library:

View: module

□ Update library list

□ Update view list

2. In the *From* section of the form, specify the source files of the Verilog modules. You can do one of the following:

   - Type the paths to the files in the *Files* field.
   - Click *Browse* and use the Browse Verilog Modules file browser to select the files one at a time.

   You can use an asterisk (*) as a wildcard in the paths you specify. For example: `/myProject/verilog/*.vams` gets modules from all files in the `verilog` directory that have a `.vams` extension and `/myproject/verilog/*` gets modules from all the files in the directory.

   Each file can contain one Verilog module or several modules.

3. In the *Library* field in the *To* section, select the library to which you want to add the Verilog modules. The drop-down list displays all the libraries that are in your library definition file (`cds.lib`).

   Note: You must have write access to the master library or its temporary directory.

4. In the *View* field, specify a name for the Verilog views. You can choose any name; the default name is `module`.

   When the Hierarchy Editor creates cellviews for the modules, it uses this name for the views. The cell names are obtained from the Verilog modules.

5. Select *Update library list* if you want the library to which you are adding Verilog modules to be added to the global library list of the configuration. If you select this option, the library is added as the first library in the library list.

   Note: If a configuration is not open, or if the library is already in the global library list of the configuration, this field is grayed-out.

6. Select *Update view list* if you want the Verilog views that you are creating to be added to the global view list of the configuration. If you select this option, the view is added as the first view in the global view list.

   Note: If a configuration is not open, or if the view is already in the global view list of the configuration, this field is grayed-out.

7. Click *OK*.

   The dialog box closes and the library is populated with Verilog modules. If there are any errors, they are displayed in the Messages section of the Hierarchy Editor.

   If a configuration is open and you chose to update the library and view lists, the updated lists are displayed in the *Global Bindings* section as well as the *Inherited Library List* and *Inherited View List* columns.

   The Verilog views are now available for binding. You can bind a cell, instance, or occurrence to the Verilog views in the same way that you bind to other views.

Compiler Options

The Hierarchy Editor uses the xmvlog compiler to create cellviews. It uses the following options with xmvlog:

```
xmvlog -use5x -work libraryName [-view viewName] [-ams]
```

where `-use5x` specifies that the Cadence library structure be created for the modules; `-work` *`libraryName`* specifies the library in which the cellviews are created; `-view` *`viewName`*, which specifies the name of the view, is used only if you fill in the *View* field in the Populate Library with Verilog Modules form (the default view name is `module`); and `-ams` is used for any source files that have a `.va` or `.vams` extension. For more information about xmvlog, see the *Cadence NC-Verilog Simulator Help*.

If the AMS plug-in is installed, the Hierarchy Editor runs xmvlog with the options specified in the *AMS – Options – Compiler* form.

If you want the xmvlog compiler to be run with any other options, specify them in the `hdl.var` file. For more information about the `hdl.var` file, see the *Cadence NC-Verilog Simulator Help*.

Referencing a Verilog File

If you want to use Verilog modules from a Verilog text file that is outside the Cadence library/cell/view structure and you cannot create cellviews for these modules in the master or explicit temporary library, you can reference the Verilog file. The file will be compiled later, by other tools in the flow, into the implicit temporary directory.

You reference a Verilog file by

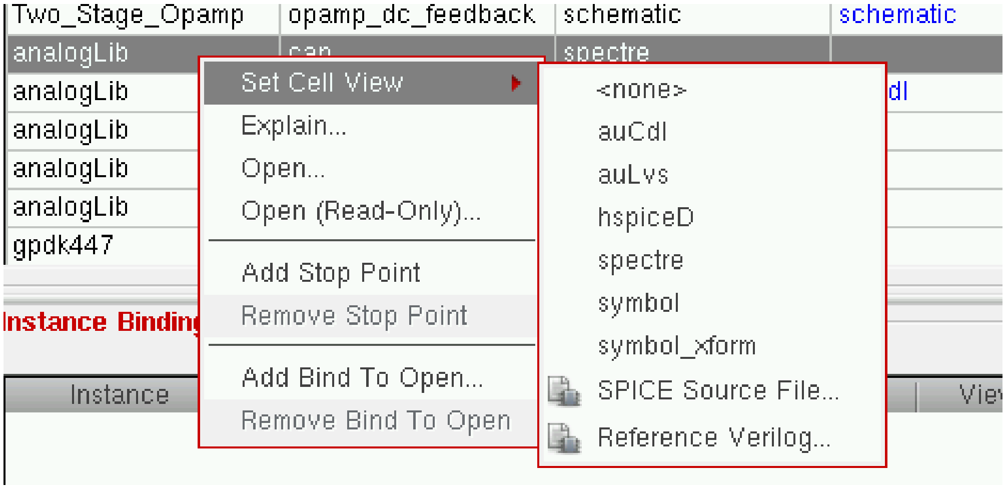
- Specifying the path to the file; and
- Binding the cell, instance, or occurrence to a view that will be later implemented with the modules in the Verilog file

This puts the `verilogfile` property on the cell, instance, or occurrence. The value of the property is the path to the Verilog file and it is stored in the `prop.cfg` file in the configuration view. The property is used by downstream processes such as the design preparation step of AMS, which reads the property and compiles the file it refers to into the implicit temporary directory.

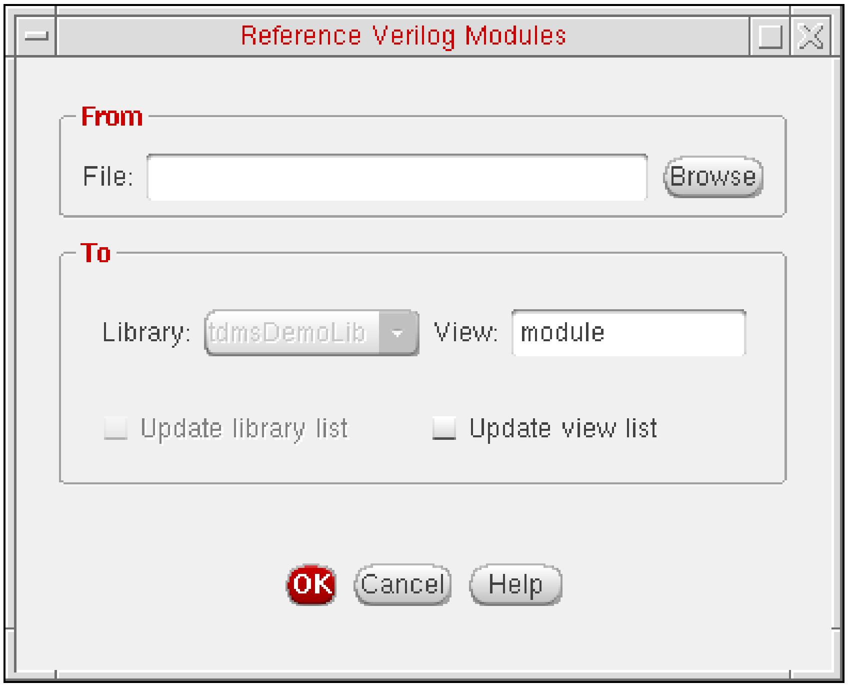
To reference a Verilog file for a cell, instance, or occurrence,

1. For a cell, display the table view of the Hierarchy Editor; for an instance, display either the table view or the tree view; for an occurrence, display the tree view and turn on the occurrence editing mode.
2. Right-click the cell, instance, or occurrence.

3. From the pop-up menu, select *Set Cell View / Set Instance View / Set Occurrence View — Reference Verilog*. For example:



The Reference Verilog Modules form appears.



4. In the form, in the *File* field, either type the path to the Verilog file that you want to use, or click *Browse* and use the Browse Verilog Modules file browser to select the file.

Note: The Verilog file you select must have one of the following extensions: `.vams` or `.v` because most Cadence applications recognize only files with these extensions as Verilog files.

5. In the *Library* field, select the library in which you want to place the cellviews when the Verilog file is compiled. The cellviews will be in the implicit temporary directory, not in the master library.

   Note: If the library is fixed by some other rules in the design data, you will not be able to edit the *Library* field. For example, design data created by the Virtuoso schematic editor typically does not allow you to change the library.

6. In the *View* field, specify a name for the Verilog view.

   The view will be later implemented with the modules in the Verilog file by other applications in the flow. For example, in the design preparation step, AMS creates cellviews for the modules in the implicit temporary directory.

   The default view name is `module`.

7. Select *Update library list* if you want the library to be added to the inherited library list of the cell, instance, or occurrence. If you select this option, the library is added as the first library in the inherited library list.

   Note: If the library is already in the inherited library list, this field is grayed-out.

8. Select *Update view list* if you want the view that you specify to be added to the inherited view list for the cell, instance, or occurrence. If you select this option, the view is added as the first view in the inherited view list.

   Note: If the view is already in the inherited view list, this field is grayed-out.

9. Click *OK*.

The cell, instance, or occurrence is bound to the view. The *View to Use* column displays the view name. The 📄 icon (file icon) next to the view name indicates that it will be implemented with a text file.

To remove a Verilog file reference,

1. If you are editing a cell, display the table view of the Hierarchy Editor; if you are editing an instance, display either the table view or the tree view; if you are editing an occurrence, display the tree view and turn on the occurrence editing mode.

2. Right-click the cell, instance, or occurrence.

3. From the pop-up menu, select *Set Cell View / Set Instance View / Set Occurrence View – <none>*.

Using Constants

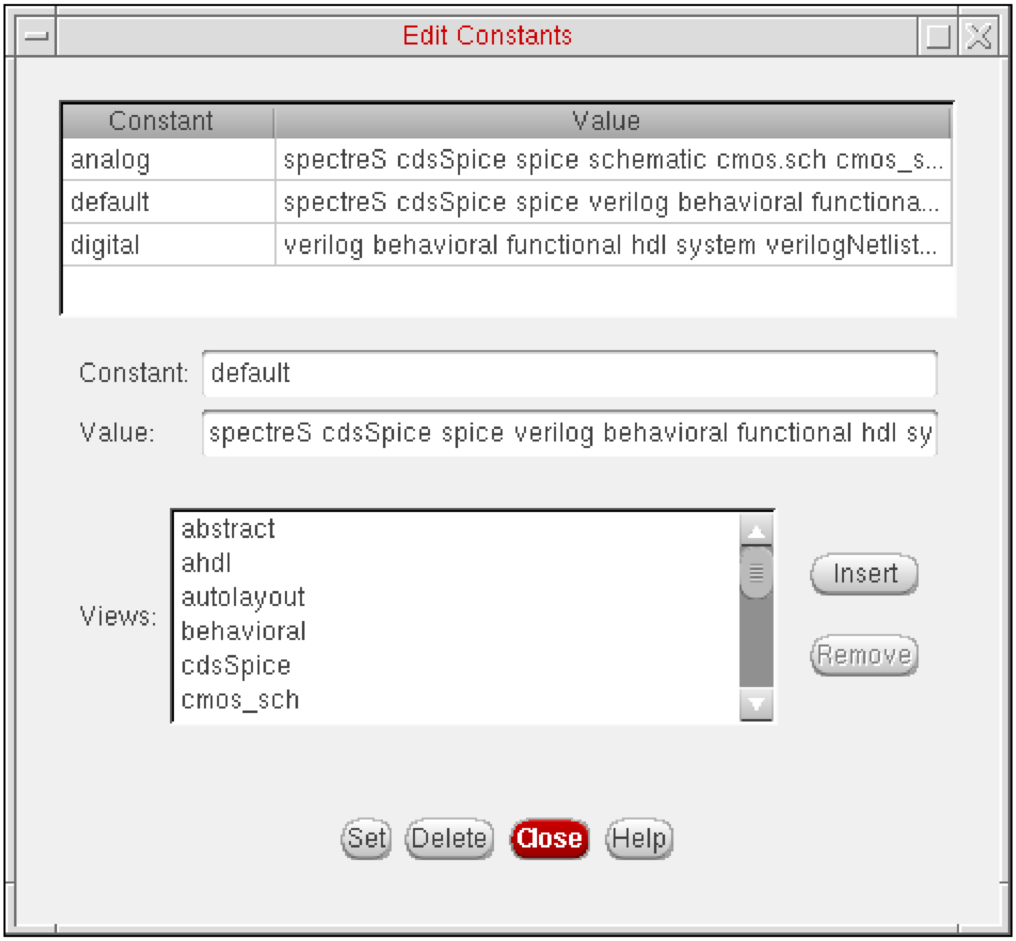
A constant is a symbolic name that represents a view list. Constants provide a way to store and retrieve long view lists. You can create a constant and use it anywhere you use a View List and a Stop List.

Creating Constants

To create a constant,

1. Choose *Edit – Constants*.

   The Edit Constants form appears.



Edit Constants

Constant	Value
analog	spectreS cdsSpice spice schematic cmos.sch cmos_s...
default	spectreS cdsSpice spice verilog behavioral functiona...
digital	verilog behavioral functional hdl system verilogNetlist...

Constant: default

Value: spectreS cdsSpice spice verilog behavioral functional hdl sy

Views:
abstract
ahdl
autolayout
behavioral
cdsSpice
cmos_sch

2. In the *Constant* field, type the name you want to use to denote the view list.

3. In the *Value* field, specify the view list by doing one of the following:

- ❑ Select a view in the *Views* list box, then click *Insert*. Repeat for all the views you want to add to the view list.
- ❑ Type the view list.

The view names must appear in order of preference. The first view that is found is used.

Note: You can change the default list of views in the *View Choices* list box. See "Changing the Views in the View Choices List Box" on page 192 for more information.

4. Click *Set*.

The Hierarchy Editor creates the constant.The constant and its value appear in the table at the top of the form.

See also "Using Constants" on page 192.

Editing Constants

To edit the views included in an existing constant,

1. Choose *Edit – Constants*.

   The Edit Constants form appears.

2. In the table at the top of the form, click the constant whose views you want to change.

   The constant name is displayed in the *Constant* field and its views are displayed in the *Value* field.

3. To add a view,

   a. In the *Value* field, place the cursor in the position that you want to add the new view.

      Views must be listed in order of preference—the first view that is found is used.

   b. In the *Views* list box, select the view that you want to add.

   c. Click *Insert*.

4. To remove a view,

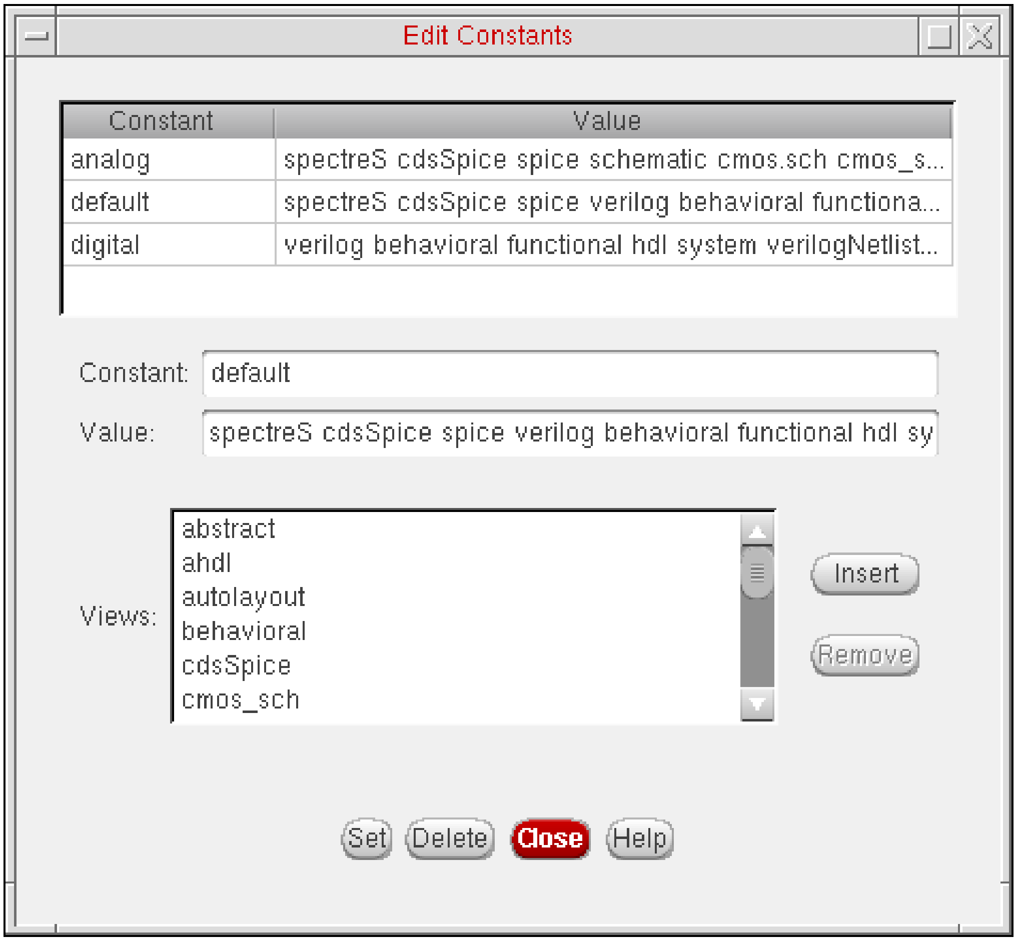
   a. In the *Value* field, double-click the view.

   b. Press the `Delete` key.

5. When you have finished editing the *Value* field, click *Set*.

See also "Using Constants" on page 192.

About the Edit Constants Form



Edit Constants

Constant	Value
analog	spectreS cdsSpice spice schematic cmos.sch cmos_s...
default	spectreS cdsSpice spice verilog behavioral functiona...
digital	verilog behavioral functional hdl system verilogNetlist...

Constant:	default
Value:	spectreS cdsSpice spice verilog behavioral functional hdl sy

Views:
abstract
ahdl
autolayout
behavioral
cdsSpice
cmos_sch

Constant is a symbolic name that represents a view list.

Value contains the list of views that the constant represents. Either type the view names or select them from the *Views* list box.

Note: Views must be listed in order of preference—the Hierarchy Editor uses the first view that is found.

Views contains the list of views from which you select views for the constant. You can customize the *View Choices* list. See “Changing the Views in the View Choices List Box” on page 192 for more information.

Insert adds the selected view to the constant.

Remove removes views from the constant.

Set assigns the views to the constant and leaves the form open.

Delete removes the selected constant name and its values.

Close closes the form and saves your new or edited constants for the current session only.

Help opens this manual.

Changing the Views in the View Choices List Box

The default list of views in the *Views* list box is defined by the `hed.constants` variable in the *`your_install_dir`*`/share/cdssetup/hierEditor/env/hed.env` file. You cannot edit this registration file but you can change the list of views by adding the variable to a local `hed.env` file.

To change the views in the *Views* list box,

1. If you do not have a `hed.env` file, create a file named `hed.env` in a directory that is in your `setup.loc` file (for example, your `$HOME` directory or current working directory).

   Note: If you have multiple `hed.env` files, the Hierarchy Editor determines which value to use based on the search order defined in the `setup.loc` file.

2. Add the following variable to the `hed.env` file:

   ```
   hed.constants viewChoices string "view1 view2 view3 view4 ..."
   ```

   where *`view1`*, *`view2`*, *`view3`*, and *`view4`* are the views that you want the *Views* list box to display. For example:

   ```
   hed.constants viewChoices string "abstract ahdl behavioral cdsSpice
   cmos_sch functional schematic spectre spectreS symbol"
   ```

For more information about the `hed.env` file, see Appendix A, “The hed.env File.”

Using Constants

To use a constant,

➤ Type the name of the constant, preceded by a dollar sign (`$`), in any view list or stop list field.

By default, the Hierarchy Editor displays the constant in the view lists or stop lists in which it is used. However, you can choose to display the expanded form of the constant (the list of views that the constant represents) instead of displaying the constant name.

To display the expanded form of the constant,

➤ Add the following variable to the `hed.env` file:

```
hed.display showConstInViewList boolean nil
```

For more information about the `hed.env` file, see Appendix A, “The hed.env File.”

Using Wildcards in a View List

You can use the asterisk character (`*`) as a wildcard in any view list—global or inherited. Use the following format to specify a wildcard in a view list:

```
[viewName ...] *
```

For example:

```
*
schematic *
schematic vhdl *
```

You cannot use `*` as part of a view name. For example, you cannot have `ns*` or `n*s` as a view in the view list.

The `*` wildcard matches any view in the cell directory. If there is more than one view, then the view that was most recently modified is picked. To determine the most recently modified view, the Hierarchy Editor compares the timestamps of the master files of each view. (The master file is the file that the `master.tag` file for the view points to.)

> **Note:** Any changes in a view that do not change the master file are not considered. For example, in the AMS Designer flow, properties associated with a cellview can change without affecting the timestamp of the master file of the view.

For example:

If the view list is ...	the following view is picked ...
`*`	The most recently modified view
`schematic *`	The `schematic` view; if the schematic view does not exist, the most recently modified view
`schematic vhdl *`	The `schematic` view; if the schematic view does not exist, the `vhdl` view; if the `vhdl` view does not exist, the most recently modified view

Changing Binding Data Color Definitions

The Hierarchy Editor displays color-coded cell and instance binding data. The following table lists the default colors:

Color	Definition
Black	Default binding
Blue	User-defined binding
Red	Error
Orange	Not in Use

Note: As you type new binding information into the Cell Bindings or Instance Bindings table, the user-defined binding color is used.

You can change the default colors by adding variables to a local `hed.env` file.

To change the default colors,

1. If you do not have a `hed.env` file, create a file named `hed.env` in a directory that is specified in your `setup.loc` file (for example, your `$HOME` directory or current working directory). You can create the file with a text editor or with the *File – Save Defaults* command.

   Note: If you have multiple `hed.env` files, the Hierarchy Editor determines which value to use based on the search order defined in the `setup.loc` file.

2. Add the following variables to the `hed.env` file:

```
hed.colorChooser bindingError string "newColor"
hed.colorChooser userBinding string "newColor"
hed.colorChooser defaultBinding string "newColor"
hed.colorChooser notInUse string "newColor"
```

   For example, if you want to change the default binding color to green, you would add the following variable:

```
hed.colorChooser defaultBinding string "green"
```

   Note: Add only those variables that you want to change from the default.

3. Save the `hed.env` file.

   Your changes are displayed the next time you start the Hierarchy Editor.

For more information about the `hed.env` file, see Appendix A, "The hed.env File."

Saving Cell Bindings Table Data to a Text File

You can save data from the *Cell Bindings* table in an ASCII file, which you can print or import into a spreadsheet or other application.

To save the *Cell Bindings* table data to a text file,

1. Customize the *Cell Bindings* table to display only the data that you want to save to a text file:

   a. Choose *View – Options* and, in the Options dialog box that appears, select the columns that you want to display and deselect the columns that you want to hide.

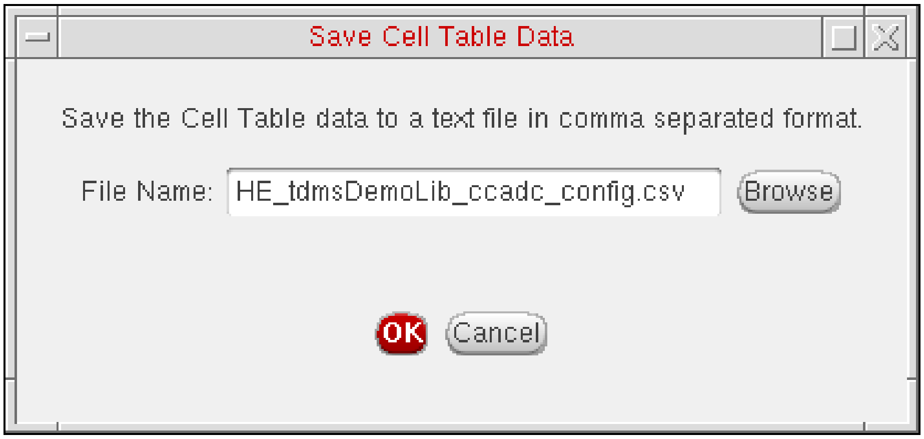
   b. Choose *View – Properties* to display or hide property columns.

   c. Choose *View – Filters* and, in the Filters form that appears, set filters to display specific cellviews.

   Only the data that is displayed will be saved.

2. Choose *File – Save Cell Table Data*.

   The Save Cell Table Data dialog box appears.



Save Cell Table Data

Save the Cell Table data to a text file in comma separated format.

File Name: HE_tdmsDemoLib_ccadc_config.csv

3. In the dialog box, type the name of the file in which to save the data or click *Browse* and use the file browser to find a directory or file.

   The default file name is `HE_`*`libName`*`_`*`cellName`*`_`*`configName`*`.csv` and the default location is the current working directory.

The *Cell Bindings* table data is saved to the file you specify. The file contains a line for each row of the table, with items separated by commas. The first line in the file lists the column headers.

For example:

```
Library,Cell,View Found,View to Use
mixSigLib,cap,behavioral,null
mixSigLib,and2,layout,behavioral
mixSigLib,counter,schematic,schematic
mixSigLib,ccadc,schematic,null
...
```

4

Setting Simulation-Control Properties

This chapter describes the following topics:

Overview of Setting Properties

The Hierarchy Editor lets you set simulation-control properties that are used by Cadence applications such as the AMS simulator in your configuration.

You can set those properties that have been registered by a Cadence application in the property dictionary. The property dictionary is a central repository of property definitions; it comprises application-specific property dictionary files located in the `your_install_dir/share/cdssetup/registry/props/` directory as well as user property dictionary files, if any (for more information about the property dictionary, see the *Cadence Application Infrastructure User Guide*). The Hierarchy Editor, by default, lets you display and set all the properties that are in the property dictionary.

You can also set properties that are not in the property dictionary by creating new property columns for them. These are typically properties that you use frequently with an application but that have not been registered in the property dictionary by the application. The Hierarchy Editor saves information about the new property columns in the configuration view in a file named `prop.cfg`.

You can set properties globally for an entire configuration or on cells, instances, or occurrences. Properties are also inherited by the hierarchy below the object on which they are set, unless the property definition specifies that it cannot be inherited.

The Hierarchy Editor stores the values of all the properties that you set in the `prop.cfg` file in the configuration view of your design. The `prop.cfg` file is read by other applications, such as simulators, that use the properties.

Displaying Properties

The Hierarchy Editor displays properties in property columns. These columns are not displayed by default.

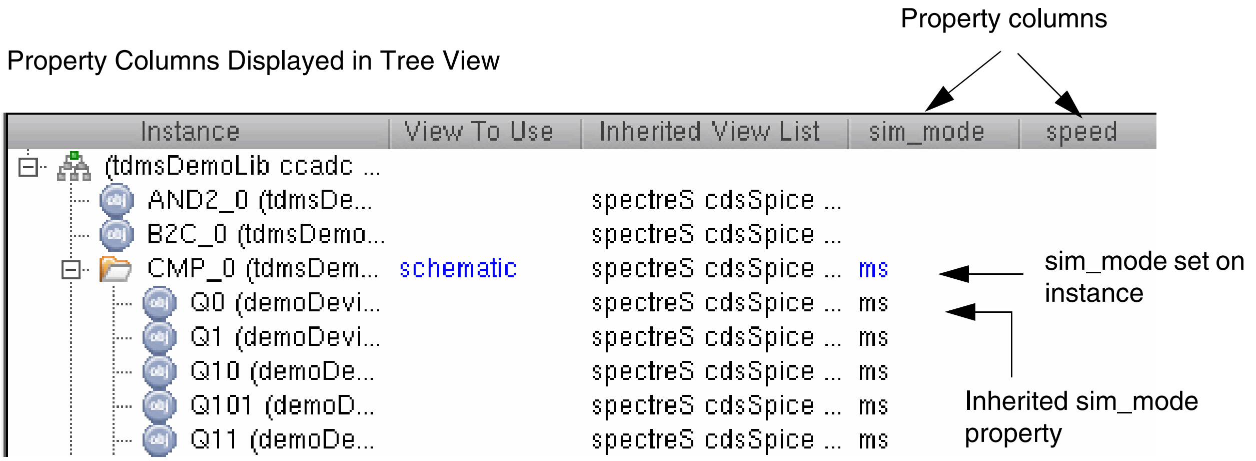
To display property columns,

1. Choose *View – Properties*.

   The Hierarchy Editor displays one column for each property that is in the property dictionary. It also displays any property columns that you added in a previous session for properties that are not in the property dictionary.

   Property columns are displayed in both the table view and the tree view. Properties that are set directly on an object are displayed in blue (the default color for user-defined values); inherited properties are displayed in black.

Property Columns Displayed in Tree View



Property columns

Instance	View To Use	Inherited View List	sim_mode	speed
(tdmsDemoLib ccadc ...				
AND2_0 (tdmsDe...		spectreS cdsSpice ...		
B2C_0 (tdmsDemo...		spectreS cdsSpice ...		
CMP_0 (tdmsDem...	schematic	spectreS cdsSpice ...	ms	
Q0 (demoDevi...		spectreS cdsSpice ...	ms	
Q1 (demoDevi...		spectreS cdsSpice ...	ms	
Q10 (demoDe...		spectreS cdsSpice ...	ms	
Q101 (demoD...		spectreS cdsSpice ...	ms	
Q11 (demoDe...		spectreS cdsSpice ...	ms	

sim_mode set on instance

Inherited sim_mode property

Note: The *View – Properties* command is not available if the `your_install_dir/share/cdssetup/registry/props/` directory or the `your_install_dir/share/cdssetup/ams` directory does not exist in your Cadence tools installation.

You can now edit any of the property columns to set properties on your design configuration; see “Setting Properties” on page 202 for more information. If you want to set a property that is not displayed, you can add a property column for it; see “Adding Property Columns” on page 209 for more information.

To display property columns by default whenever the Hierarchy Editor is started,

1. Choose *View – Properties*.
2. Choose *File – Save Defaults*.

3. In the Save Defaults dialog box, specify the directory in which to save your default settings, then click *OK*.

   See "Saving Custom Settings" on page 78 for more information about saving defaults.

To hide all property columns,

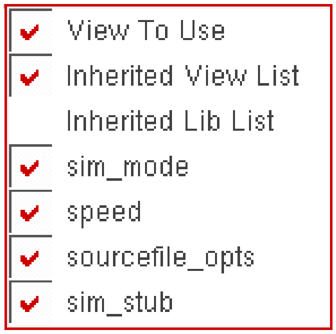
1. Choose *View – Properties*.

   Property columns are hidden.

To hide or display individual property columns,

1. Right-click the property column heading.

   A pop-up menu, listing all the columns that can be displayed, appears. The columns that are currently displayed have a check-mark next to them.



✓ View To Use
✓ Inherited View List
Inherited Lib List
✓ sim_mode
✓ speed
✓ sourcefile_opts
✓ sim_stub

2. Select the property columns that you want to display and deselect the ones that you want to hide.

   The tree view or table view display is updated to reflect your choices. These settings remain in effect for the session only and are not saved in the `hed.env` file when you save your defaults.

To sort the cells in the *Cell Bindings* or *Instance Bindings* table by property,

1. Click the heading of the property column.

To display a description of the property,

1. Place your cursor over the property column heading.

   A description of the property is displayed. If the property column has a drop-down list of value choices, you can also place your cursor over each value to see its description.

   Note: A description of the property and its values is displayed only if it was provided as part of the property definition.

You can also resize property columns and change the order in which they appear. For more information, see Resizing Columns.

Setting Properties

You can set properties on your design globally so that they apply to your entire design configuration, or on cells, instances, or occurrences.

A global property applies to the entire configuration, a cell property applies to all the instances of the cell, an instance property applies to a single instantiation of a cell, and an occurrence property applies to a specific path.

Properties are also inherited by the hierarchy below the object on which they are set, unless a property definition specifies that it cannot be inherited.

Since a property can be set on an object at multiple levels, a certain order of precedence determines the property's value. The Hierarchy Editor uses the following precedence (listed from highest to lowest):

- Occurrence property
- Instance property
- Cell property
- Inherited property
- Global property

Setting Properties on a Global Basis

You can set properties on a global basis by setting them on the root cellview of the configuration. Global properties become the default for the configuration and apply to every level of the hierarchy. Global properties can be overridden at the cell, instance, and occurrence levels.

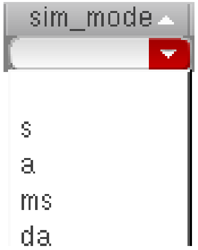
To set a property on a global basis,

1. In the tree view or the table view of the Hierarchy Editor, choose *View – Properties*.

   Property columns are displayed for all the properties that are defined in the property dictionary or in the `prop.cfg` file.

2. In the root cellview row (the root cellview is identified by a green pyramid icon in the Information column), click the column of the property you want to set and either type in the value or select it from the drop-down list.

Properties of type `enum` have a drop-down list of values from which you select a value. Properties of other types, such as integer or string, have an editable field. For example:



The property value that you set is displayed in the property column of all objects in the hierarchy, unless it is overriden by other values set at the cell, instance, or occurrence level. The value is displayed in blue (the default color for user-defined values) for the root cellview and in black for all other objects.

Note: If a property's definition specifies that it cannot be inherited, then it is not inherited by the hierarchy below it even if it is set at the global level.

Setting Properties on a Cell

You can also set properties on a cell. A cell-level property applies to every instantiation of the cell. It is also inherited by all objects below the cell in the hierarchy, unless its definition specifies that it cannot be inherited.

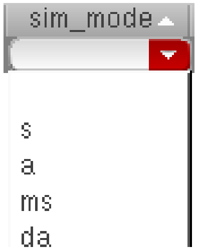
Cell-level properties override global properties and can be overriden at the instance and occurrence levels.

To set a property on a cell

1. Choose *View – Parts Table* to display the table view of the Hierarchy Editor.
2. Choose *View – Properties*.

   Property columns are displayed for all the properties that are defined in the property dictionary or in the `prop.cfg` file.
3. In the cell row, click the column of the property you want to set and either type in the value or select it from the drop-down list.

Properties of type `enum` have a drop-down list of values from which you select a value. Properties of other types, such as integer or string, have an editable field. For example:



The property value appears in the property column of the cell in the Cell Bindings section. Because the property is also inherited by the hierarchy below the cell, the property value also appears in the property columns of all the cells that are under that cell. (A property will not be inherited by the hierarchy below the cell if the property definition specifies that it cannot be inherited.)

To see a list of the instantiations to which the property applies,

1. In the *Cell Bindings* section, right-click the cell.
2. From the pop-up menu, select *Explain*.

   The Explain dialog box appears. The Instantiations section lists the instantiations of the cell.

To see the hierarchy below the cell to which the property also applies,

1. Choose *View – Tree* to display the tree view of the configuration.
2. Double-click the cell or click its expansion icon.

Setting Properties on an Instance

You can also set properties on an instance. Instance properties apply to only one instantiation of a cell (unlike cell properties that apply to all the instantiations of a cell). Note, however, that instance properties can apply to multiple objects—if the cell that contains the instance is used in multiple places in the design, the property applies to the instance in all those locations.

Instance properties are also inherited by all objects in the hierarchy below the instance, unless a property's definition specifies that it cannot be inherited.

Instance-level properties override properties set at the global or cell level and can be overriden at the occurrence level.

Setting Instance Properties from the Tree View

To set a property on an instance from the tree view of the Hierarchy Editor,

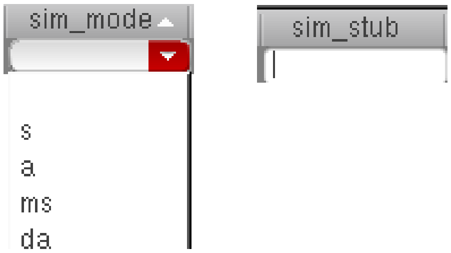
1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.
2. If the *Occurrence Mode* button on the toolbar is selected, deselect it.

   The tree view now displays *Target: Instance*.

   Note: If an instance has already been identified as an occurrence, the tree view will display *Target: Occurrence* when it is selected, and any property you set on it will be added to the occurrence, not the instance.
3. Choose *View – Properties*.

   Property columns are displayed for all the properties that are defined in the property dictionary or in the `prop.cfg` file.
4. In the *Tree View* section, click the property column of the instance on which you want to set the property and either type in the value or select it from the drop-down list.

   Properties of type `enum` have a drop-down list of values from which you select a value. Properties of other types, such as integer or string, have an editable field. For example:



Setting Instance Properties from the Table View

To set a property on an instance from the table view,

1. Choose *View – Parts Table* and *View – Instance Table* to display the table view of the Hierarchy Editor, if it is not already displayed.
2. Choose *View – Properties*.

   Property columns are displayed for all the properties that are defined in the property dictionary or in the `prop.cfg` file.

3. In the *Cell Bindings* section, select the cell that contains the instance on which you want to set the property.

   The *Instance Bindings* section displays all the instances the cell contains.

4. In the *Instance Bindings* section, click in the property column of the instance on which you want to set the property and either type in the value or select it from the drop-down list.

   Properties of type `enum` have a drop-down list of values from which you select a value. Properties of other types, such as integer or string, have an editable field.

   Note: You cannot add an instance property to an object that has already been defined as an occurrence (identified by a icon). To do so, you must remove the occurrence binding first.

To see which objects the property applies to,

1. In the *Cell Bindings* section in the table view, right-click the cell that contains the instance on which you set the property.

2. From the pop-up menu, select *Explain*.

   The Explain dialog box appears. The Instantiations section lists the instantiations of the cell. The property will apply to the instance in all these instantiations of its parent cell.

To see the hierarchy below the instance to which the property also applies,

1. Choose *View – Tree* to display the tree view of the configuration.

2. Double-click the instance or click its expansion icon.

Setting Properties on an Occurrence

You can also set properties on an occurrence. An occurrence property is set on a specific path (see About Occurrences for more information about occurrences). It is also inherited by all objects in the hierarchy below the occurrence, unless the property definition specifies that it cannot be inherited.

An occurrence property has the highest precedence; it cannot be overriden by a property set at any other level of the hierarchy.

You can only add an occurrence property in the tree view of the Hierarchy Editor; you cannot add it in the table view.

To set an occurrence property,

1. Choose *View – Tree* to display the tree view of the configuration, if it is not already displayed.

2. Turn on the occurrence editing mode by clicking the *Set bindings...on Occurrence* button in the toolbar:

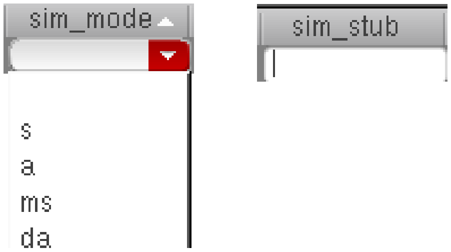
   The tree view displays *Target: Occurrence*.

3. Choose *View – Properties*, if necessary.

   Property columns are displayed for all the properties that are defined in the property dictionary or in the `prop.cfg` file.

4. In the *Tree View* section, click in the property column of the occurrence on which you want to set the property, and either type in the value or select it from the drop-down list.

   Properties of type `enum` have a drop-down list of values from which you select a value. Properties of other types, such as integer or string, have an editable field. For example:



The property value appears in the property column with the icon (occurrence icon) next to it. The path is now identified uniquely as an occurrence; if any other attribute is set on it, such as a stop point, it will be set on the occurrence, not the instance.

Removing Properties

You can remove a property in two ways:

- By removing it from the object on which it is set
- By removing it from the entire design configuration

Removing Properties from Specific Objects

You can remove a property from a specific object on which it is set—that is, from a cell, instance, or occurrence.

When you remove a property from an object, it is also removed from all the objects in the hierarchy below it that inherited the property.

To remove a property from a specific object,

1. If you are removing a property from a cell, display the table view of the Hierarchy Editor; if you are removing it from an instance, display either the table view or the tree view; if you are removing it from an occurrence, display the tree view and turn on the occurrence editing mode.
2. In the object's row, click in the property column of the property that you want to remove.
3. Either delete the text in the field, or, if a drop-down list appears, set the value to the empty space at the top of the list.

Note: You can only remove a property from objects on which it is set directly; you cannot remove it from objects that have inherited the property.

Removing Properties from the Entire Design Configuration

To remove a property from all objects on which it is set, that is to remove all references to it from the design configuration, you delete its property column.

You can delete only those property columns that you have added with the *Edit – Add Property Column* command. You cannot delete property columns of properties that are defined in the property dictionary.

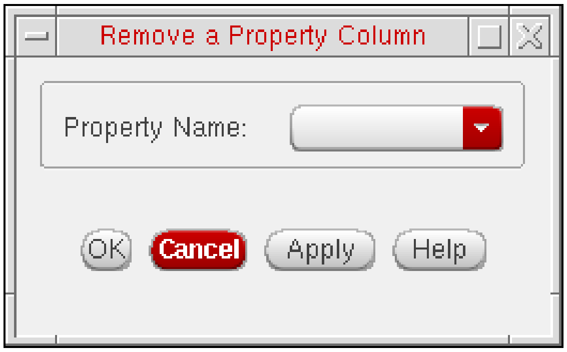
Caution

When you remove a property column, you are removing that property from all objects on which it is set. You will not be able to undo the delete command. Therefore, remove a column only if you are sure that you want to remove the property entirely from your design.

To delete a property column,

1. Choose *Edit – Remove Property Column*.

   The Remove a Property column dialog box appears.



2. In the dialog box, select the property column that you want to delete.

   The list only displays those property columns that you have added with the *Edit – Add Property Column* command.

3. Click *OK*.

Adding Property Columns

The Hierarchy Editor, by default, lets you display and set all the simulation-control properties that are in the property dictionary. If you want to set a property that is not in the property dictionary, you can do so by creating a property column for it. You typically add properties that you use frequently with an application.

Note: You should only add simulation-control properties that are understood by a Cadence application.

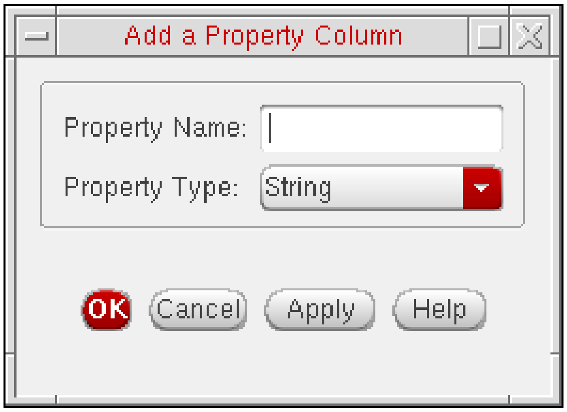
When you create a new property column, set the property on objects in your design, and then save the configuration, the new property column definition is added to the `prop.cfg` file in the configuration view. If a `prop.cfg` file does not exist, it is created. (For information about the `prop.cfg` file, see “About the prop.cfg File” on page 211.)

The next time that you display properties with the *View – Properties* command, the property column that you added is always displayed.

To add a property column,

1. Choose *Edit – Add Property Column*.

   The Add a Property Column dialog box appears.



   Note: The *Edit – Add Property Column* command is only available if the *View – Properties* command is selected.

2. In the *Property Name* field, specify the name of the property for which you want to add a property column.

   Note: The name must match the property name that the application recognizes.

3. In the *Property Type* field, select the property type from the pulldown menu. The following choices are available: *String*, *Int*, *Double*.

   The Hierarchy Editor uses the property type you specify to do type-checking when the property is set.

4. Click *Apply* to display the new property column or *OK* to display the new property column and close the dialog box.

   The new property column is displayed.

5. Set the new property on objects in your design configuration. (See "Setting Properties" on page 202 for more information.)

6. Choose *File – Save* to save your configuration.

Note: If you create a new property column for a property but do not set the property on any objects in your configuration, the new property column definition is not saved when you save the configuration. For the new definition to be saved, the property must be set on at least one object in your configuration.

About the prop.cfg File

The `prop.cfg` file is a property file created by the Hierarchy Editor in the configuration view. The `prop.cfg` file stores the values of all properties that you set with the Hierarchy Editor. It also stores information about any property columns that you add.

The `prop.cfg` file is read by applications that use the properties.

Caution

Do not edit the prop.cfg file manually.

5

Using Plug-Ins

This chapter covers the following topics:

- About Plug-ins on page 214
- Loading Plug-In Applications on page 215
- Introducing Pin Checker on page 220
- Removing Plug-In Applications on page 245

About Plug-ins

Plug-ins are applications that are added to the Hierarchy Editor and that are used from the Hierarchy Editor user interface. Some Cadence products have been added to the Hierarchy Editor as plug-ins.

Note: You cannot add any other Cadence products, or your own applications, as plug-ins.

The Plug-Ins Menu

If at least one plug-in has been added to the Hierarchy Editor, the Hierarchy Editor menu bar displays the *Plugins* menu. If the *Plugins* menu is not present, no plug-ins are available with the Hierarchy Editor.

Loading Plug-In Applications

The *Plugins* menu contains all the applications that are available from the Hierarchy Editor. To use any application listed in the menu, you must first load it. The Hierarchy Editor does not load the plug-ins by default.

When you load a plug-in application, it appears in the Hierarchy Editor, typically as a menu on the menu bar or an icon on the tool bar. You can load a plug-in either from the command line when you start the Hierarchy Editor or from the Hierarchy Editor user interface.

Note: Only those Cadence applications that are predefined as Hierarchy Editor plug-ins are displayed in the *Plugins* menu. You cannot add any other Cadence applications, or your own menu items, to the menu.

Loading a Plug-In from the Hierarchy Editor User Interface

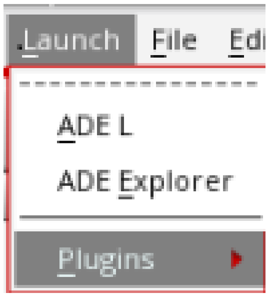
To load a plug-in application from the Hierarchy Editor user interface,

1. Start the Hierarchy Editor.

   The *Plugins* sub-menu appears in the *Launch* menu. The menu lists the applications that you can load. A check mark next to a menu item indicates the application is already loaded.

   Note: The *Plugins* sub-menu appears only if plug-ins have been added to the Hierarchy Editor.

2. Choose *Launch – Plugins* and select a plugin that you want to load, for example, *AMS HDL Settings*.



   The application appears in the Hierarchy Editor as specified by the plug-in. For instance, it could appear as an additional menu in the Hierarchy Editor menu bar or an icon in the Hierarchy Editor tool bar.

3. (Optional) If you want the Hierarchy Editor to automatically load the plug-in application every time it starts, save the Hierarchy Editor defaults with the *File – Save Defaults* command.

Loading a Plug-in from the Command Line

To load a plug-in application from the command line,

1. Start the Hierarchy Editor with the following command:

   ```
   cdsHierEditor -plugin pluginName [pluginOptions]
   ```

   To load more than one plug-in, use the following format:

   ```
   cdsHierEditor -plugin pluginName [pluginOptions] -plugin pluginName
   [pluginOptions] ...
   ```

   Note: `-plugin` must be the last argument specified with the `cdsHierEditor` command. It can be followed only by other `-plugin` arguments.

   When the Hierarchy Editor starts, the plug-in application appears in the Hierarchy Editor graphical user interface, typically as a menu in the menu bar or an icon in the tool bar.

2. (Optional) If you want the Hierarchy Editor to automatically load the plug-in application every time it starts, save the Hierarchy Editor defaults with the *File – Save Defaults* command.

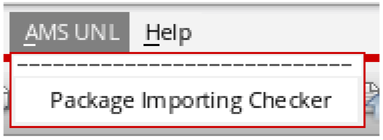
Checking Imported Packages

The *Package Importing Checker* functionality allows you to view a list of imported packages, comprising VHDL and SystemVerilog (SV) modules and their dependency relationships. In addition, it enables you to find errors and rectify them during simulation before netlisting a design. This, in turn, helps in reducing the errors that are generated during netlisting and saves time.

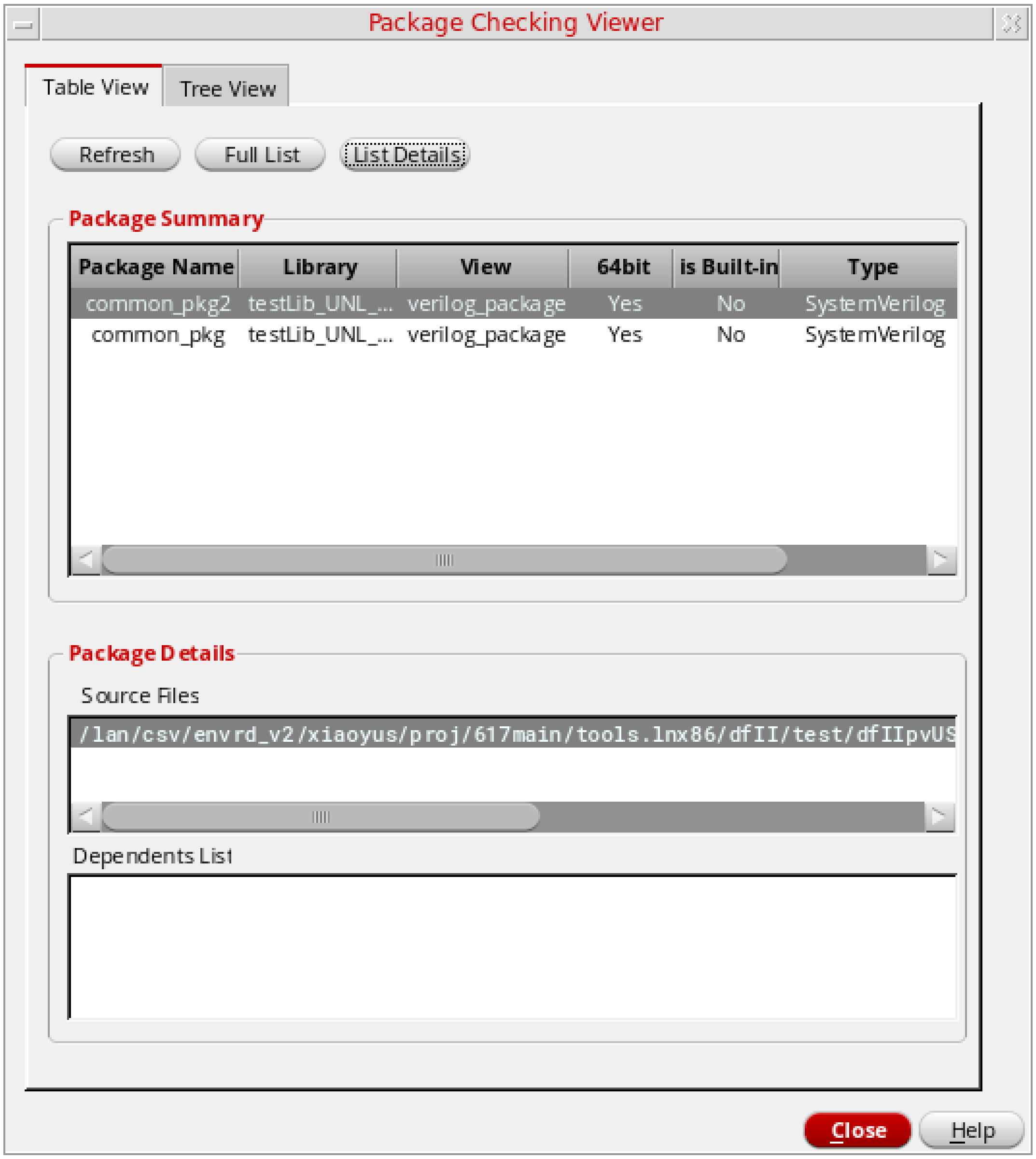
To view the list of imported packages using the built-in AMS UNL plugin available in HED, perform the following steps:

1. To enable AMS UNL plugin, choose *Launch – Plugins – AMS HDL Settings* from the Virtuoso Hierarchy Editor window.

   Once the plugin is enabled, the *AMS UNL* menu is added to the HED menu.



2. To view the list of imported packages, choose *AMS UNL – Package Importing Checker*. The Package Checking Viewer form is displayed.



Package Name	Library	View	64bit	is Built-in	Type
common_pkg2	testLib_UNL_...	verilog_package	Yes	No	SystemVerilog
common_pkg	testLib_UNL_...	verilog_package	Yes	No	SystemVerilog

```
/lan/csv/envrd_v2/xiaoyus/proj/617main/tools.lnx86/dfII/test/dfIIpvUS
```

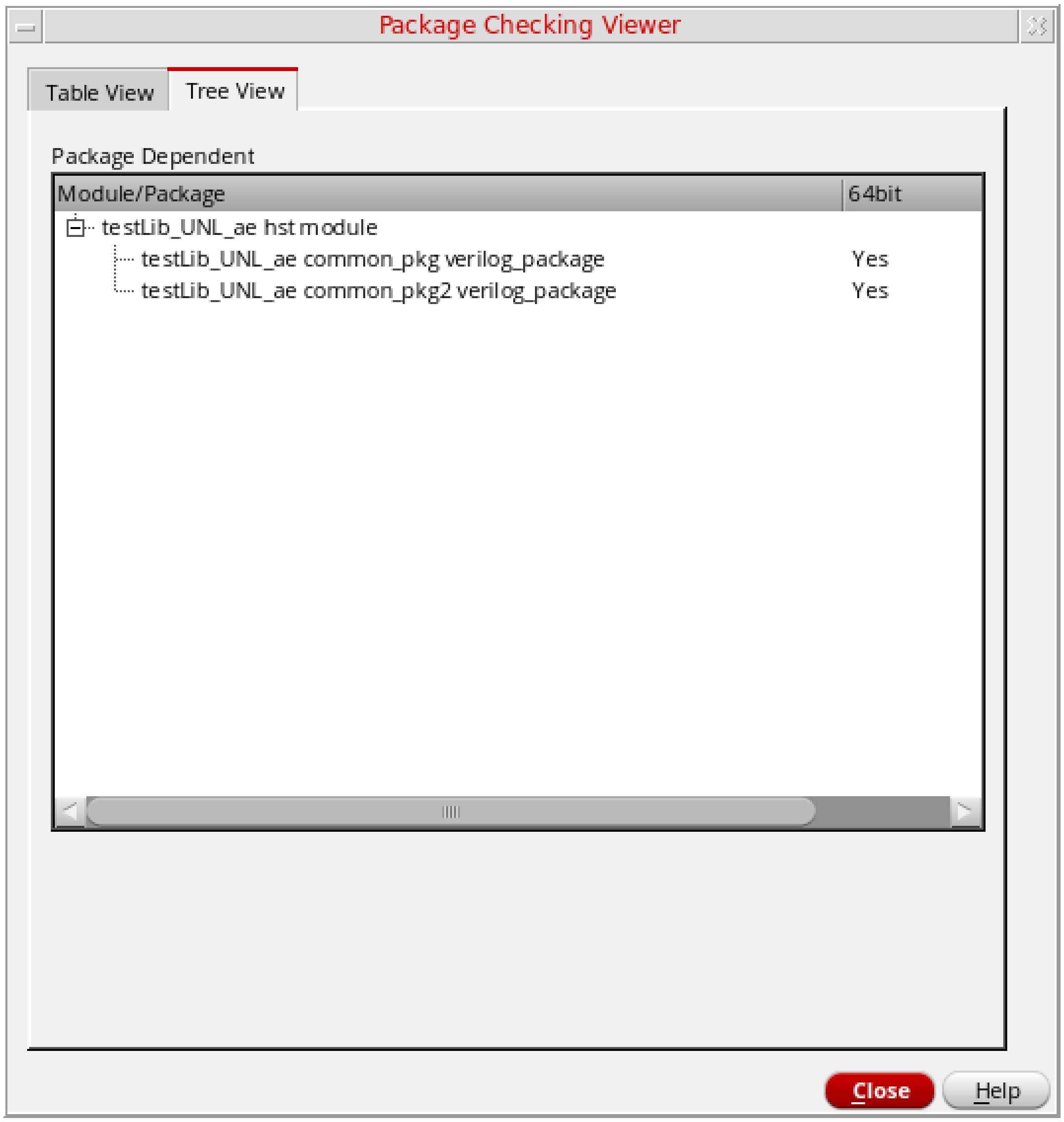
The components of the *Table View* tab on the Package Checking Viewer form are:

- *Refresh*: Refreshes the list of packages in the *Package Summary* area.
- *Full List*: Displays the built-in packages in the *Package Summary* area.

  Note: The source file and package dependents are not displayed in the *Package Summary* area.

- *List Details*: Displays the associated source files log and dependents' list of the selected package.
- *Package Summary*: Displays the list of packages.
- *Package Details*: Displays the source file and dependents' list of the selected package.

When you click the *Tree View* tab on the Package Checking Viewer form, a list of packages and their dependency relationships are displayed, as shown below.



Module/Package	64bit
testLib_UNL_ae hst module	
testLib_UNL_ae common_pkg verilog_package	Yes
testLib_UNL_ae common_pkg2 verilog_package	Yes

Introducing Pin Checker

Pin Checker is a tool that checks the connections between an instance and it's master in the config design. A config design can consist of both text and non-text cellviews.

Pin Checker helps you to find the mismatch in the connectivity and allows you to fix the issues before netlisting. This, in turn, helps in reducing the errors that are generated during netlisting and saves time.

It performs the following checks:

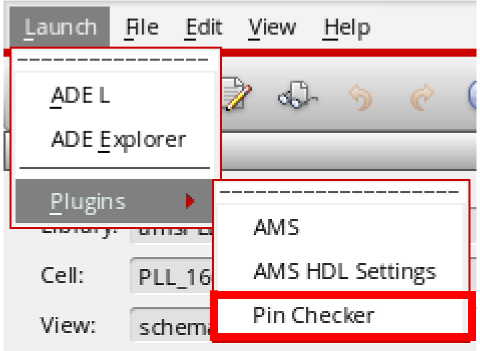
- If the parent containing the instance is text, then it checks the connectivity of the instance and it's switch master. It compares the number and size of terminals in the switch master with the instance.
- If the parent containing the instance is schematic, then it performs the following checks:
    - ❑ Checks the connectivity of the instance and it's place master. It compares the number of terminals in the place master with the instance.
    - ❑ Checks the connectivity of instance's place master and switch master. It compares the number and direction of terminals between the switch master and the place master.

Important

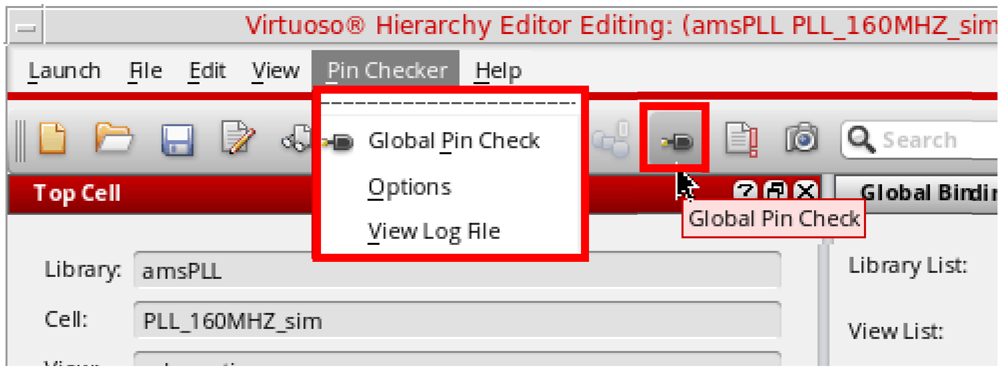
Pin Checker supports the hierarchy of text and non-text views. The support for text cellview is available for `VerilogD`, `VerilogAMS`, `SystemVerilog`, `VHDL`, and `VHDLAMS`.

Enabling Pin Checker

To enable Pin Checker, select *Launch — Plugins — Pin Checker* from the Virtuoso Hierarchy Editor window.



Once the plugin is enabled, the *Pin Checker* menu is added to the HED menu items. In addition, the *Global Pin Check* button is also displayed on the HED toolbar.



For more information on Virtuoso Hierarchy Editor, see Virtuoso Hierarchy Editor Overview.

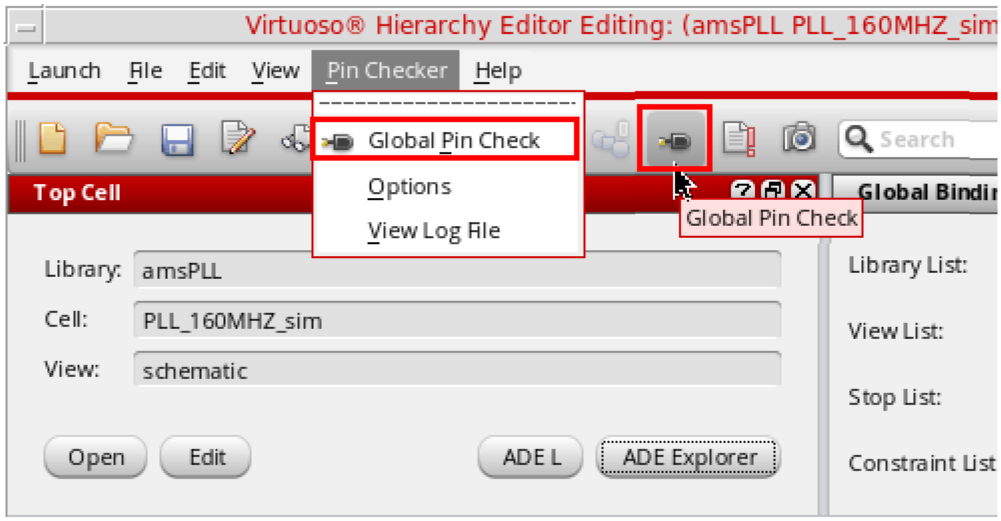
Modes of Pin Checking

- Global Pin Check
- Instance Pin Check
- Cell Instantiation Pin Check

Global Pin Check

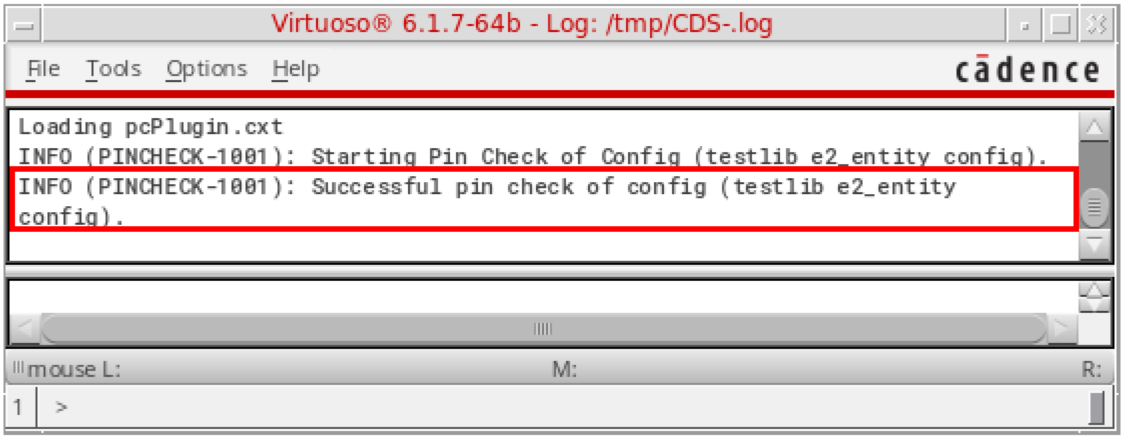
You can perform hierarchical pin check on all the instances in a design. To do this, you need to perform one of the following steps:

- Select *Global Pin Check* from the *Pin Checker* menu.
- Click the *Global Pin Check* button on the HED toolbar.



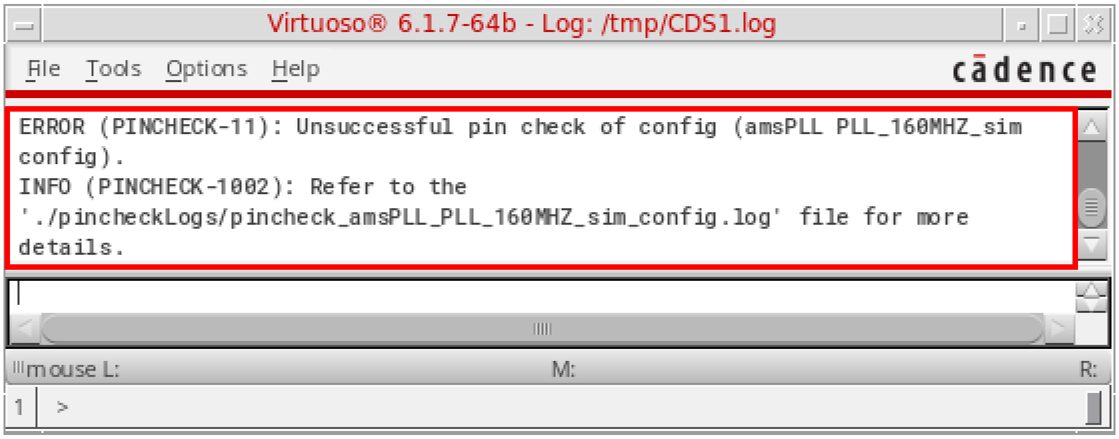
Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

After the pin checking operation on a design is successfully completed without any error or warning, a successful pin check message is displayed in CIW.



```
Loading pcPlugin.cxt
INFO (PINCHECK-1001): Starting Pin Check of Config (testlib e2_entity config).
INFO (PINCHECK-1001): Successful pin check of config (testlib e2_entity
config).
```

If there is any error or warning in the design, the unsuccessful pin check message is displayed in CIW and the pin check log file is displayed.



```
ERROR (PINCHECK-11): Unsuccessful pin check of config (amsPLL PLL_160MHZ_sim
config).
INFO (PINCHECK-1002): Refer to the
'./pincheckLogs/pincheck_amsPLL_PLL_160MHZ_sim_config.log' file for more
details.
```

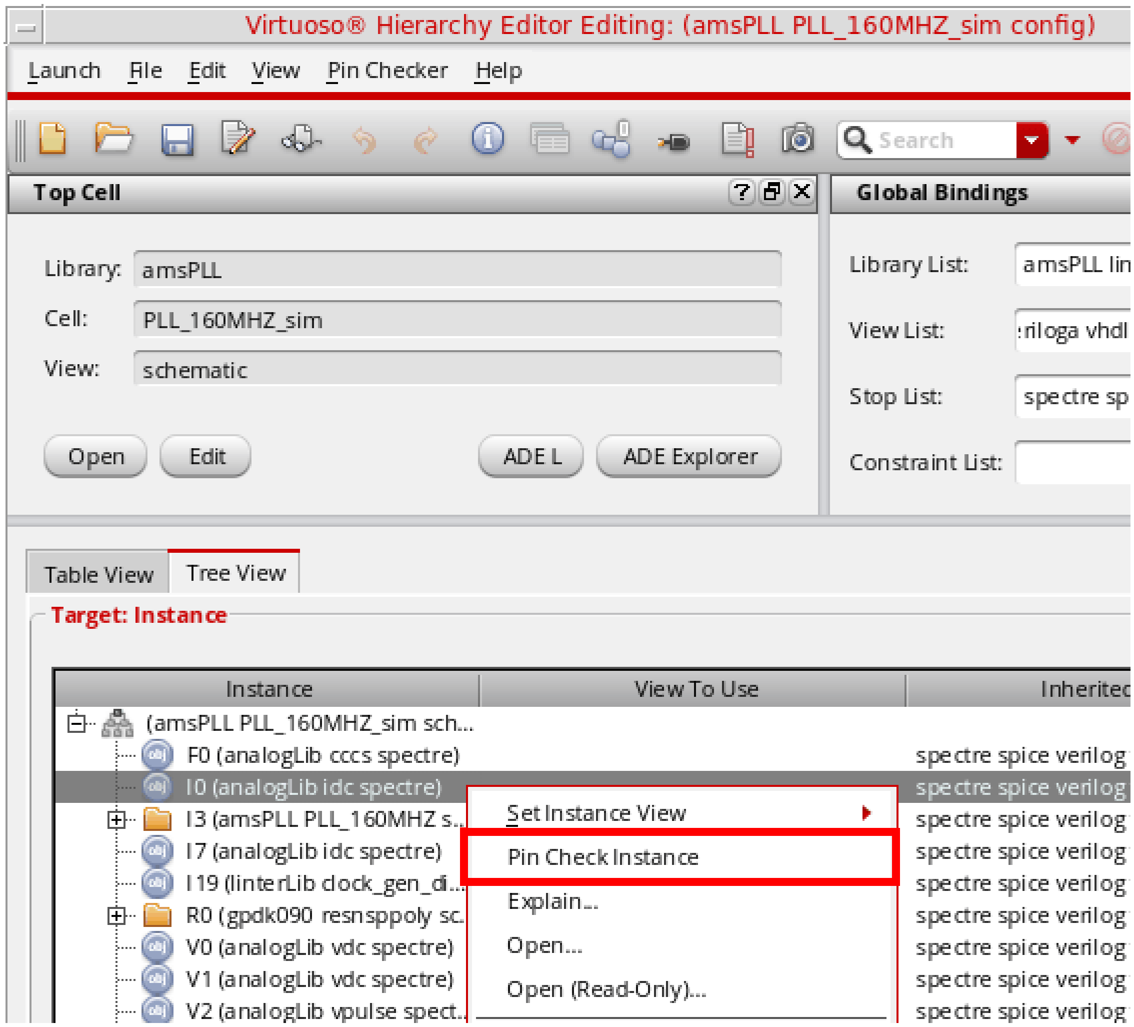
For more information on the Pin Check log file, see Log File Creation.

Instance Pin Check

- Single Instance Pin Check
- Multiple Instances Pin Check

Single Instance Pin Check

To perform pin checking on a single instance, in the *Tree View* of the Virtuoso Hierarchy Editor window, right-click the instance and select *Pin Check Instance* from the context-sensitive menu.

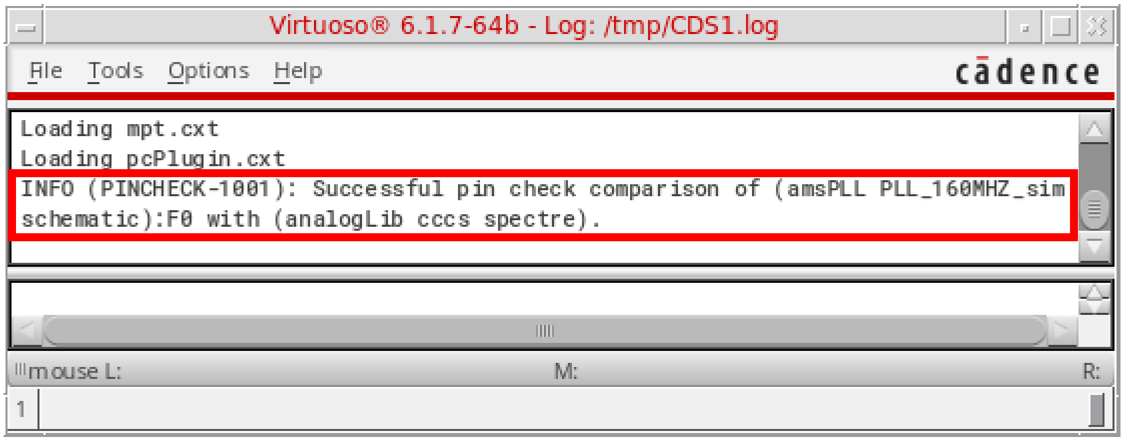


Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

Library List: amsPLL lin
View List: :riloga vhdl
Stop List: spectre sp
Constraint List:

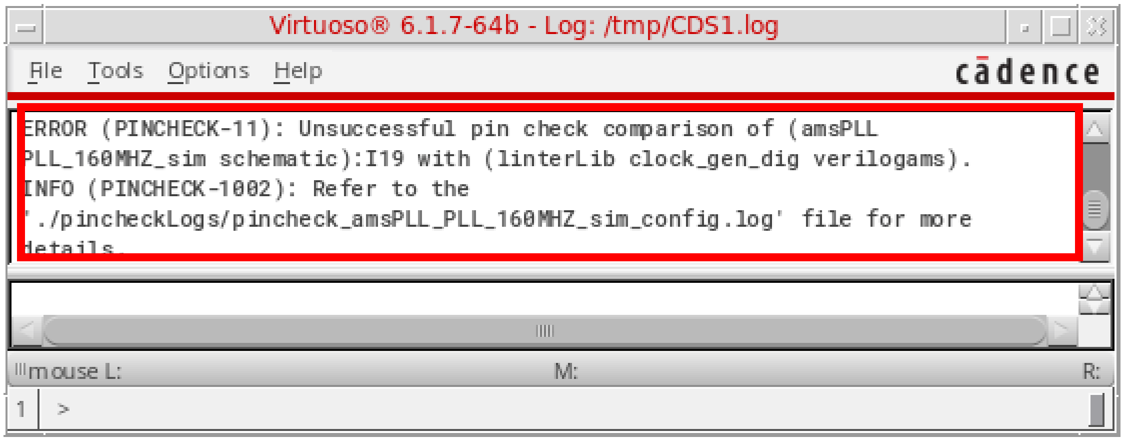
Instance	View To Use	Inheritec
(amsPLL PLL_160MHZ_sim sch...		
F0 (analogLib cccs spectre)		spectre spice verilog
I0 (analogLib idc spectre)		spectre spice verilog
I3 (amsPLL PLL_160MHZ s...		spectre spice verilog
I7 (analogLib idc spectre)		spectre spice verilog
I19 (linterLib clock_gen_di...		spectre spice verilog
R0 (gpdk090 resnsppoly sc...		spectre spice verilog
V0 (analogLib vdc spectre)		spectre spice verilog
V1 (analogLib vdc spectre)		spectre spice verilog
V2 (analogLib vpulse spect...		spectre spice verilog

After pin checking is successfully completed without any error or warning on the selected instance, the successful pin check message is displayed in CIW.



```
Loading mpt.cxt
Loading pcPlugin.cxt
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_sim
schematic):F0 with (analogLib cccs spectre).
```

If there is any error or warning in the design, the unsuccessful pin check message is displayed in CIW and the Pin Check log file is displayed.

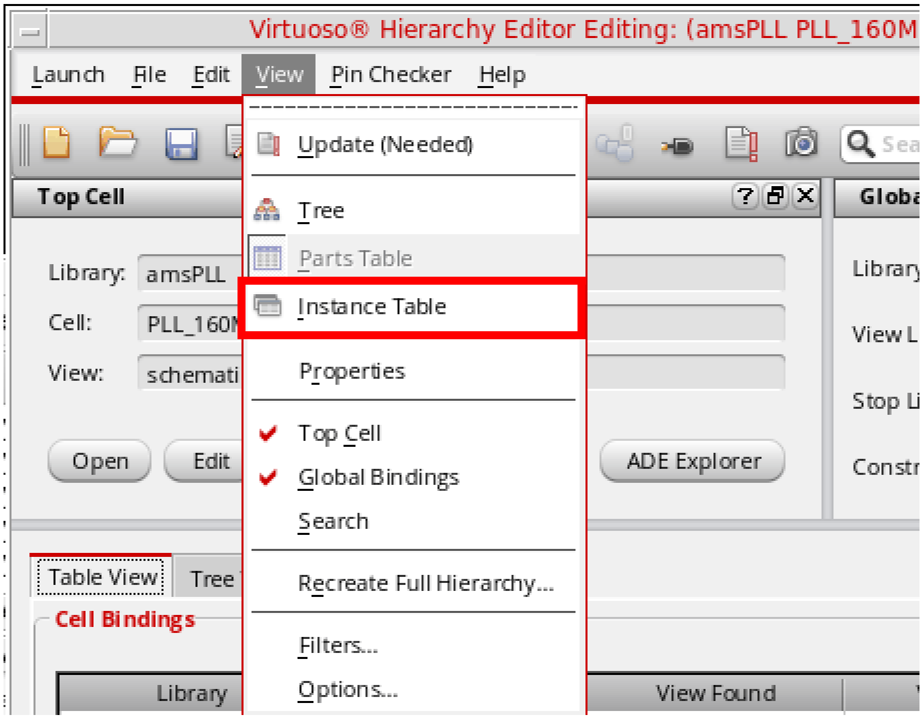


```
ERROR (PINCHECK-11): Unsuccessful pin check comparison of (amsPLL
PLL_160MHZ_sim schematic):I19 with (linterLib clock_gen_dig verilogams).
INFO (PINCHECK-1002): Refer to the
'./pincheckLogs/pincheck_amsPLL_PLL_160MHZ_sim_config.log' file for more
details.
```

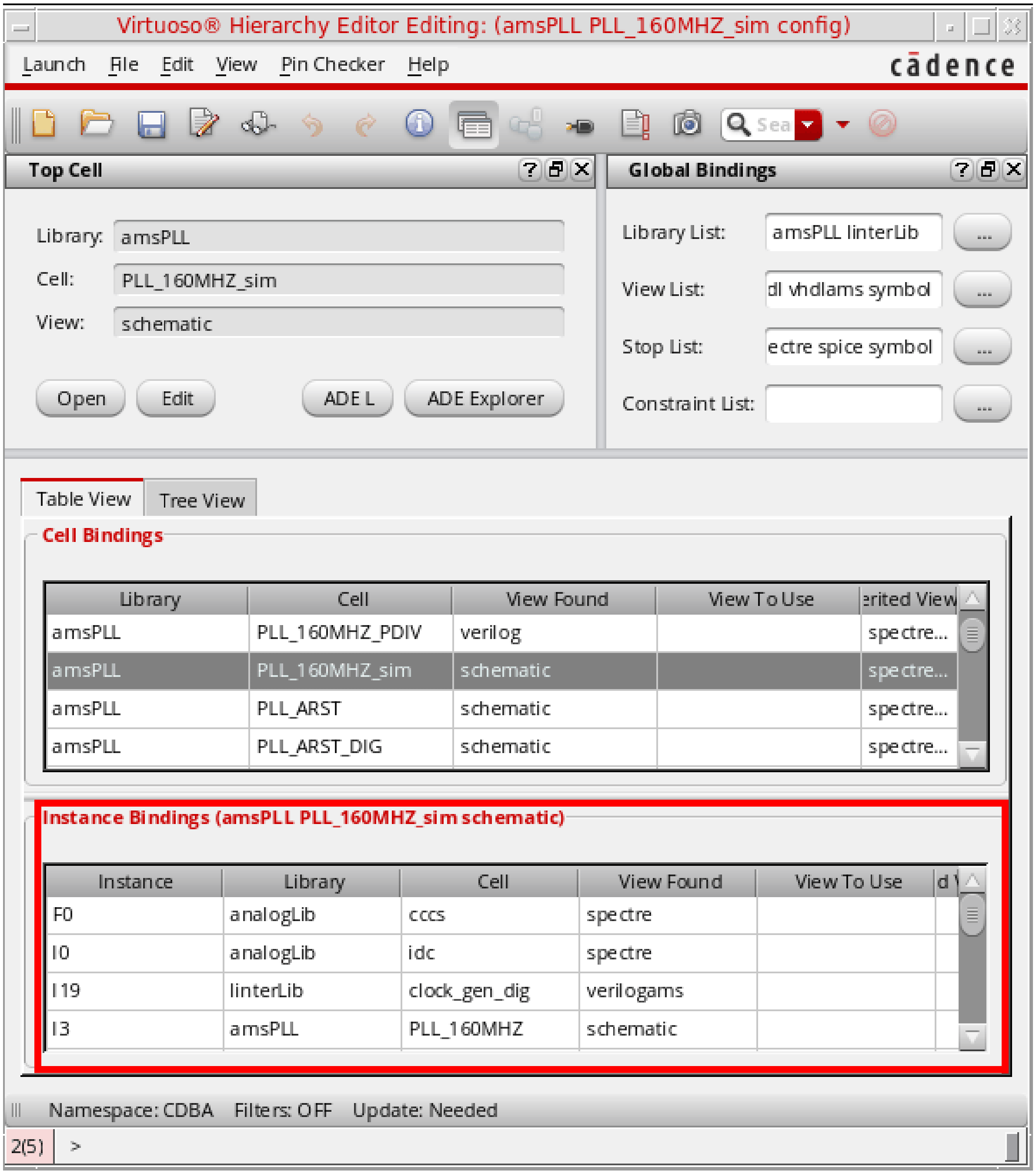
For more information on the Pin Check log file, see Log File Creation.

Apart from this, you can perform pin checking on a particular instance of the selected cell. To do this, you need to perform the following steps:

1. Open the *Instance Bindings* section in the *Tree View* by selecting *View – Instance Table* from the Virtuoso Hierarchy Editor window.



2. The *Instance Bindings* section is displayed in the *Table View* that lists all the instances of the cell, which is selected in the *Cell Bindings* section.



Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

Library List: amsPLL linterLib
View List: dl vhdlams symbol
Stop List: ectre spice symbol
Constraint List:

Cell Bindings

Library	Cell	View Found	View To Use	erited View
amsPLL	PLL_160MHZ_PDIV	verilog		spectre...
amsPLL	PLL_160MHZ_sim	schematic		spectre...
amsPLL	PLL_ARST	schematic		spectre...
amsPLL	PLL_ARST_DIG	schematic		spectre...

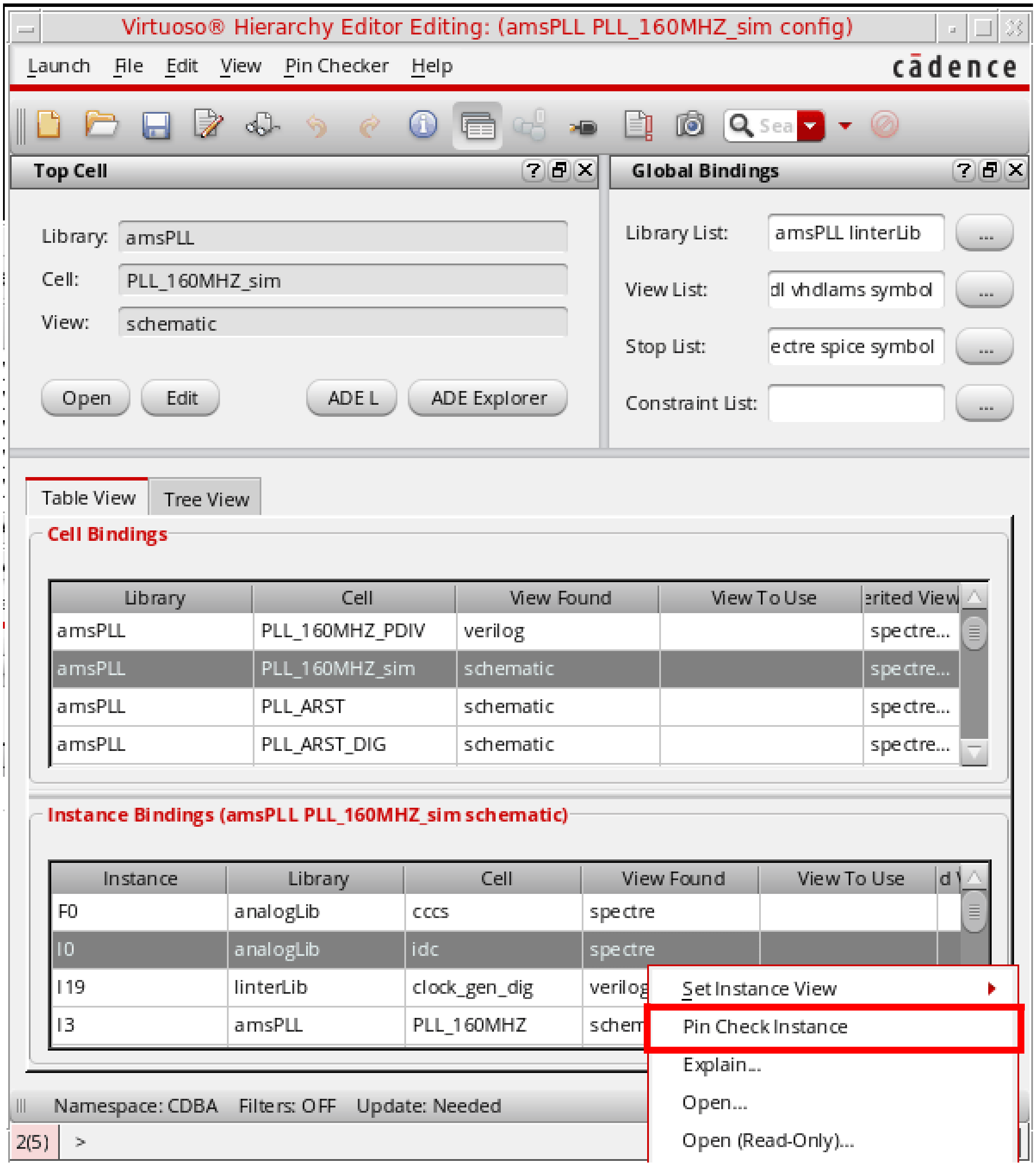
Instance Bindings (amsPLL PLL_160MHZ_sim schematic)

Instance	Library	Cell	View Found	View To Use	d \
F0	analogLib	cccs	spectre		
I0	analogLib	idc	spectre		
I19	linterLib	clock_gen_dig	verilogams		
I3	amsPLL	PLL_160MHZ	schematic		

Namespace: CDBA Filters: OFF Update: Needed

2(5) >

3. To perform pin checking on a single instance of the selected cell, right-click the instance in the *Instance Bindings* section and select *Pin Check Instance* from the context-sensitive menu.



Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

Library List: amsPLL linterLib
View List: dl vhdlams symbol
Stop List: ectre spice symbol
Constraint List:

Cell Bindings

Library	Cell	View Found	View To Use	erited View
amsPLL	PLL_160MHZ_PDIV	verilog		spectre...
amsPLL	PLL_160MHZ_sim	schematic		spectre...
amsPLL	PLL_ARST	schematic		spectre...
amsPLL	PLL_ARST_DIG	schematic		spectre...

Instance Bindings (amsPLL PLL_160MHZ_sim schematic)

Instance	Library	Cell	View Found	View To Use	d \
F0	analogLib	cccs	spectre		
I0	analogLib	idc	spectre		
I19	linterLib	clock_gen_dig	verilog		
I3	amsPLL	PLL_160MHZ	schem		

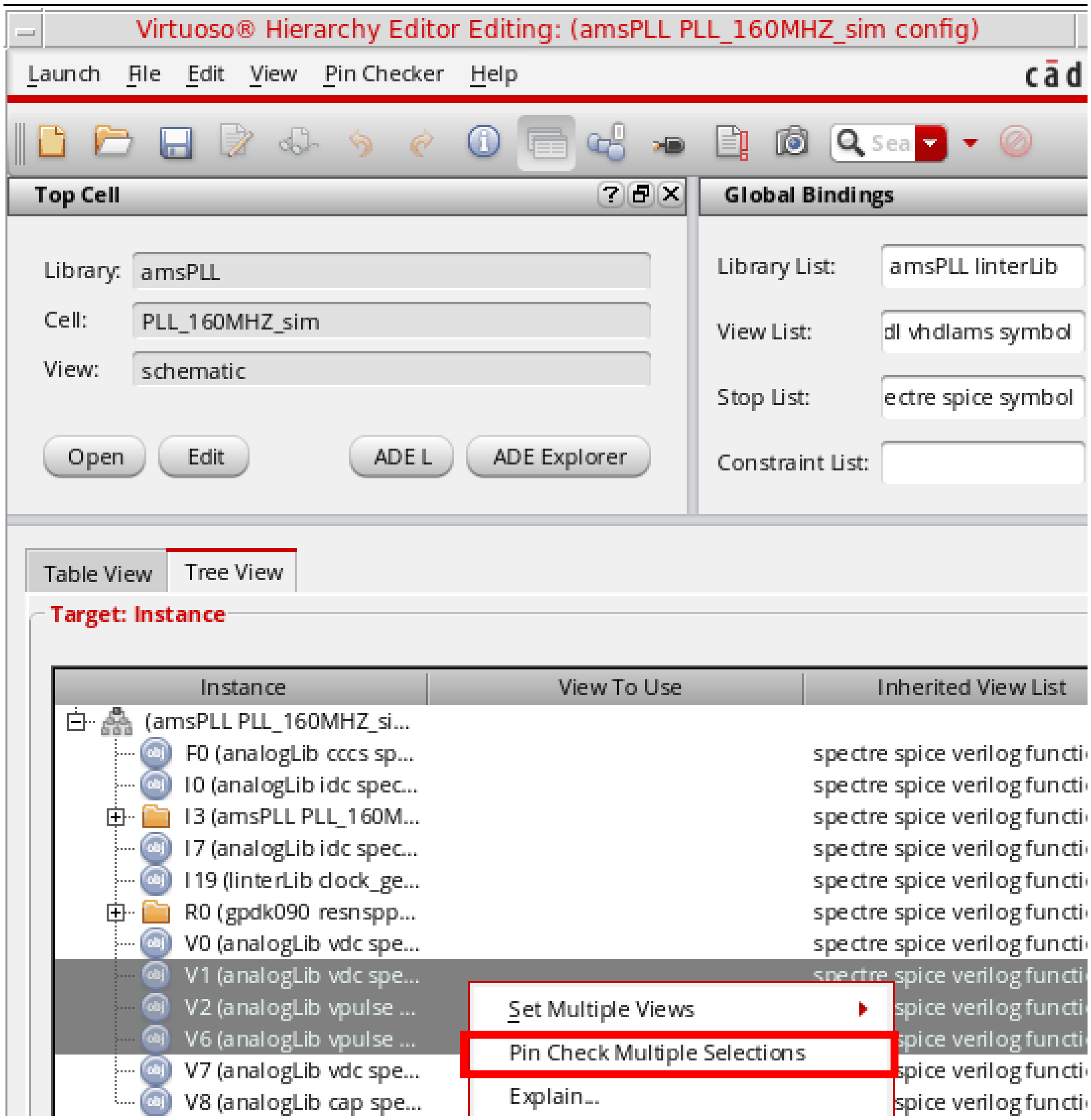
Namespace: CDBA Filters: OFF Update: Needed

2(5) >

Multiple Instances Pin Check

You can also select multiple instances for pin checking. To do this, you need to perform the following steps:

1. In the *Tree View* of the Virtuoso Hierarchy Editor window, select multiple instances by using the `Ctrl` or `Shift` key with the left mouse button.
2. Right-click the selected instances and choose *Pin Check Multiple Selections* from the context-sensitive menu.

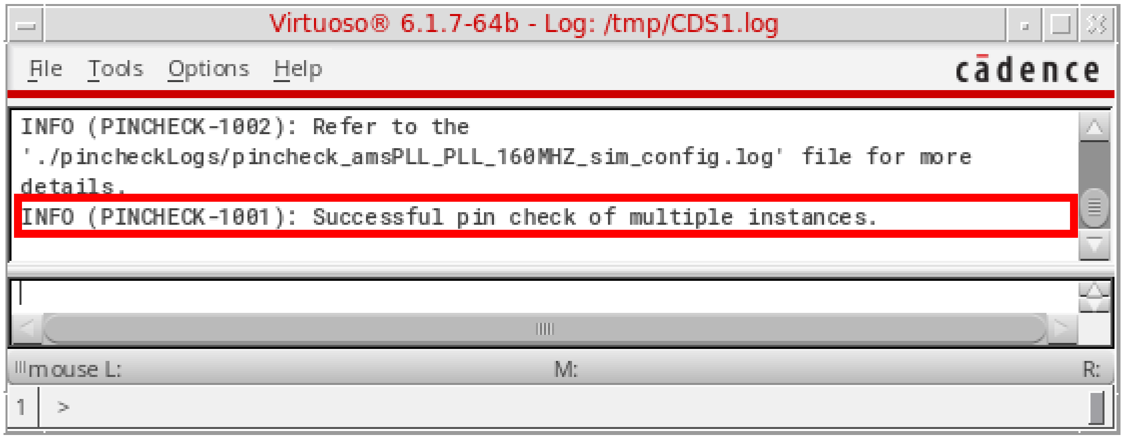


Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

Library List: amsPLL linterLib
View List: dl vhdlams symbol
Stop List: ectre spice symbol
Constraint List:

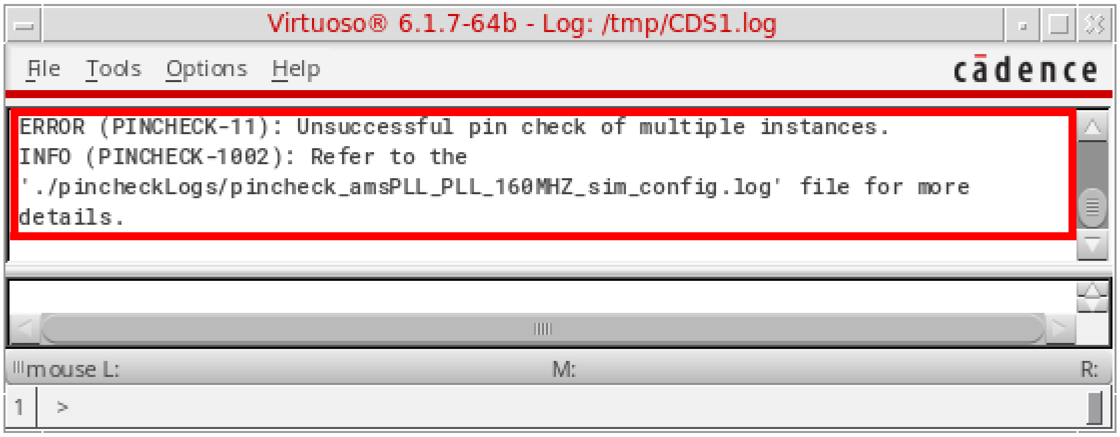
Instance	View To Use	Inherited View List
(amsPLL PLL_160MHZ_si...		
F0 (analogLib cccs sp...		spectre spice verilog functi
I0 (analogLib idc spec...		spectre spice verilog functi
I3 (amsPLL PLL_160M...		spectre spice verilog functi
I7 (analogLib idc spec...		spectre spice verilog functi
I19 (linterLib clock_ge...		spectre spice verilog functi
R0 (gpdk090 resnspp...		spectre spice verilog functi
V0 (analogLib vdc spe...		spectre spice verilog functi
V1 (analogLib vdc spe...		spectre spice verilog functi
V2 (analogLib vpulse ...		spice verilog functi
V6 (analogLib vpulse ...		spice verilog functi
V7 (analogLib vdc spe...		spice verilog functi
V8 (analogLib cap spe...		spice verilog functi

After pin checking is successfully completed without any error or warning on multiple instances, the successful pin check message is displayed in CIW.



```
INFO (PINCHECK-1002): Refer to the
'./pincheckLogs/pincheck_amsPLL_PLL_160MHZ_sim_config.log' file for more
details.
INFO (PINCHECK-1001): Successful pin check of multiple instances.
```

If there is any error or warning in the design, the unsuccessful pin check message is displayed in CIW and the Pin Check log file is displayed.

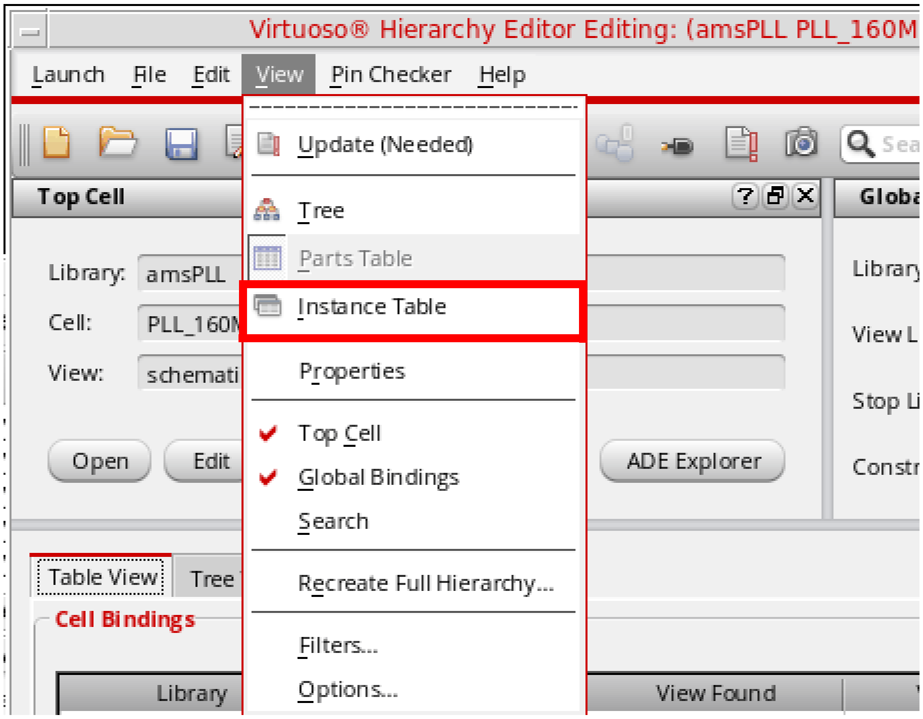


```
ERROR (PINCHECK-11): Unsuccessful pin check of multiple instances.
INFO (PINCHECK-1002): Refer to the
'./pincheckLogs/pincheck_amsPLL_PLL_160MHZ_sim_config.log' file for more
details.
```

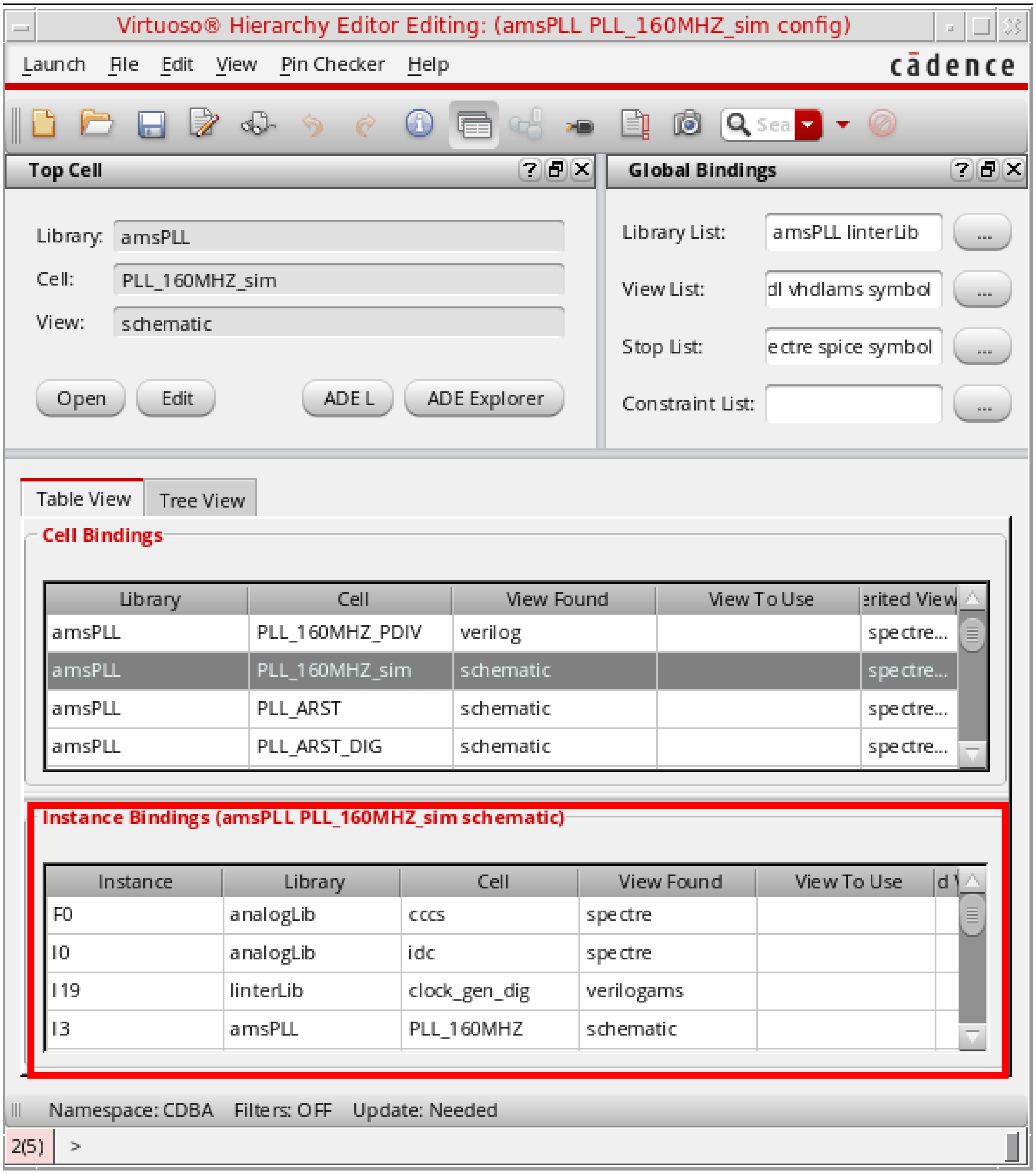
For more information on the Pin Check log file, see Log File Creation.

Apart from this, you can perform pin checking on multiple instances of the selected cell. To do this, you need to perform the following steps:

1. Open the *Instance Bindings* section in the *Tree View* by selecting *View – Instance Table* from the Virtuoso Hierarchy Editor window.



2. The *Instance Bindings* section is displayed in the *Table View* that lists all the instances of the cell, which is selected in the *Cell Bindings* section.



Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

Library List: amsPLL linterLib
View List: dl vhdlams symbol
Stop List: ectre spice symbol
Constraint List:

Cell Bindings

Library	Cell	View Found	View To Use	erited View
amsPLL	PLL_160MHZ_PDIV	verilog		spectre...
amsPLL	PLL_160MHZ_sim	schematic		spectre...
amsPLL	PLL_ARST	schematic		spectre...
amsPLL	PLL_ARST_DIG	schematic		spectre...

Instance Bindings (amsPLL PLL_160MHZ_sim schematic)

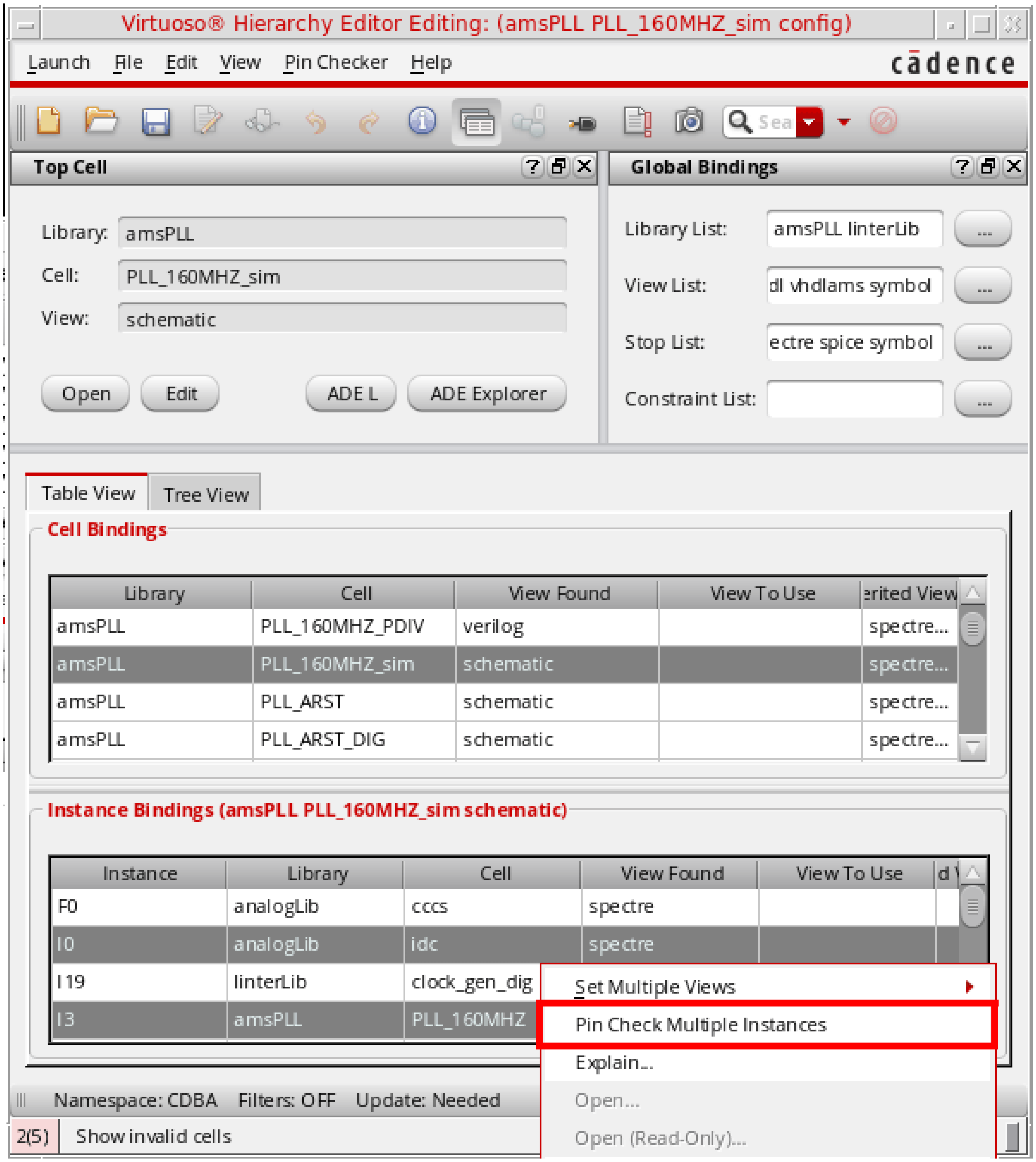
Instance	Library	Cell	View Found	View To Use	d \
F0	analogLib	cccs	spectre		
I0	analogLib	idc	spectre		
I19	linterLib	clock_gen_dig	verilogams		
I3	amsPLL	PLL_160MHZ	schematic		

Namespace: CDBA Filters: OFF Update: Needed

2(5) >

3. To perform pin checking on multiple instances of the selected cell, select multiple instances from the *Instance Bindings* section by using the `Ctrl` or `Shift` key with the left mouse button.

4. Right-click the selected instances and select *Pin Check Multiple Instances* from the context-sensitive menu.



Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

Library List: amsPLL linterLib
View List: dl vhdlams symbol
Stop List: ectre spice symbol
Constraint List:

Cell Bindings

Library	Cell	View Found	View To Use	erited View
amsPLL	PLL_160MHZ_PDIV	verilog		spectre...
amsPLL	PLL_160MHZ_sim	schematic		spectre...
amsPLL	PLL_ARST	schematic		spectre...
amsPLL	PLL_ARST_DIG	schematic		spectre...

Instance Bindings (amsPLL PLL_160MHZ_sim schematic)

Instance	Library	Cell	View Found	View To Use	d \
F0	analogLib	cccs	spectre		
I0	analogLib	idc	spectre		
I19	linterLib	clock_gen_dig			
I3	amsPLL	PLL_160MHZ			

Namespace: CDBA Filters: OFF Update: Needed

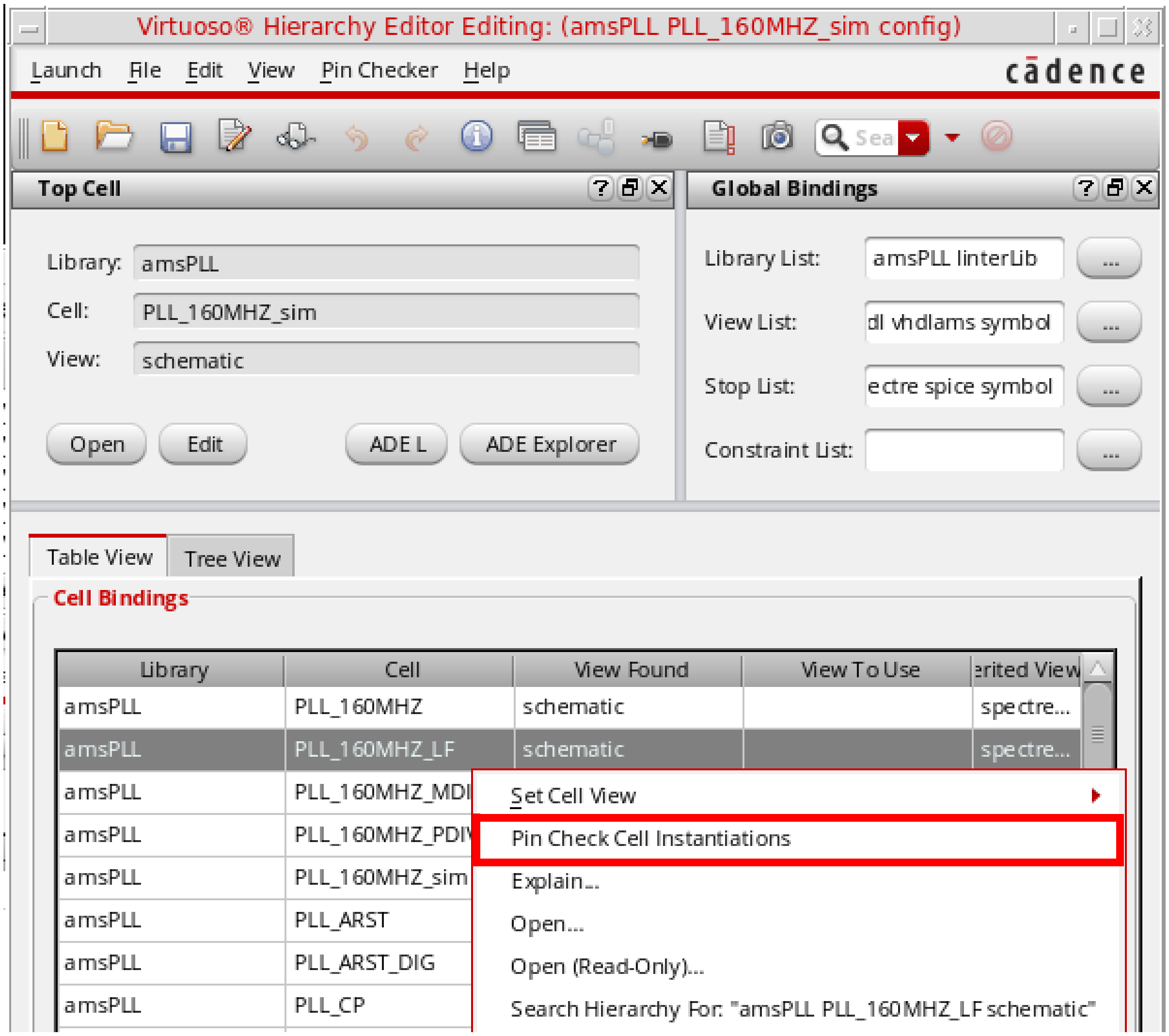
2(5) Show invalid cells

Cell Instantiation Pin Check

- Single Cell Instantiation Pin Check
- Multiple Cell Instantiations Pin Check

Single Cell Instantiation Pin Check

You can perform pin checking on a particular cell at all the places where it is instantiated in a design. To do this, in the *Table view* of the Virtuoso Hierarchy Editor window, right-click the *Cell Bindings* row and select *Pin Check Cell Instantiations* from the context-sensitive menu.

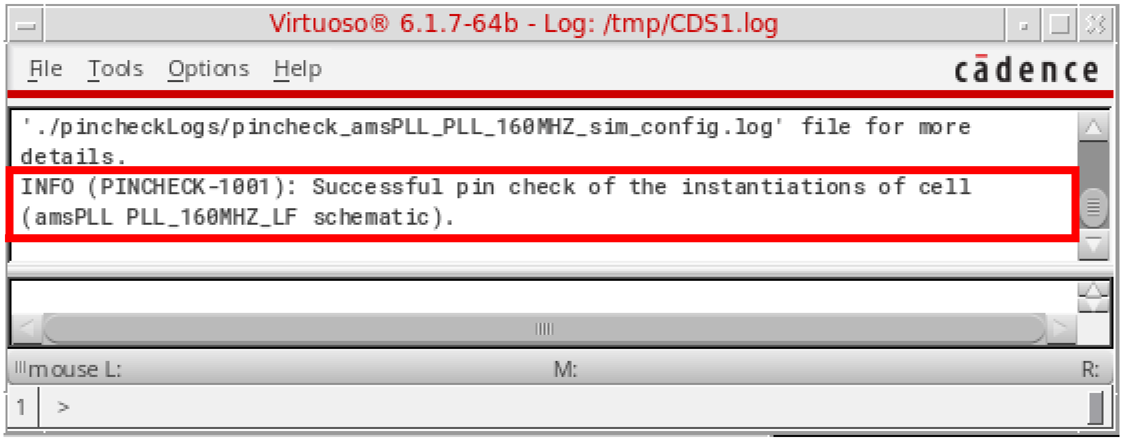


Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

Library List: amsPLL linterLib
View List: dl vhdlams symbol
Stop List: ectre spice symbol
Constraint List:

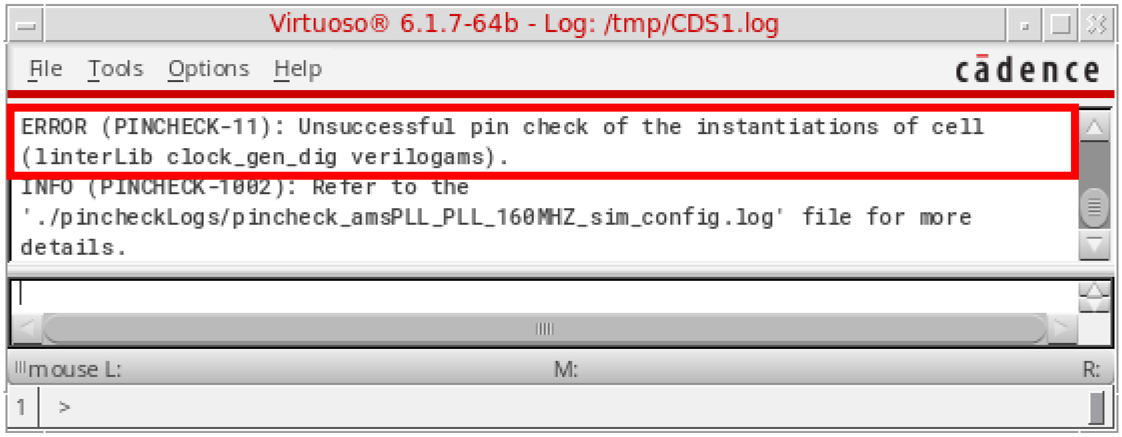
Library	Cell	View Found	View To Use	erited View
amsPLL	PLL_160MHZ	schematic		spectre...
amsPLL	PLL_160MHZ_LF	schematic		spectre...
amsPLL	PLL_160MHZ_MDI			
amsPLL	PLL_160MHZ_PDI\			
amsPLL	PLL_160MHZ_sim			
amsPLL	PLL_ARST			
amsPLL	PLL_ARST_DIG			
amsPLL	PLL_CP			

After cell instantiation is successfully completed without any error or warning on all the instances, the successful pin check message is displayed in CIW.



```
'./pincheckLogs/pincheck_amsPLL_PLL_160MHZ_sim_config.log' file for more
details.
INFO (PINCHECK-1001): Successful pin check of the instantiations of cell
(amsPLL PLL_160MHZ_LF schematic).
```

If there is any error or warning in the design, the unsuccessful pin check message is displayed in CIW and the Pin Check log file is displayed.



```
ERROR (PINCHECK-11): Unsuccessful pin check of the instantiations of cell
(linterLib clock_gen_dig verilogams).
INFO (PINCHECK-1002): Refer to the
'./pincheckLogs/pincheck_amsPLL_PLL_160MHZ_sim_config.log' file for more
details.
```

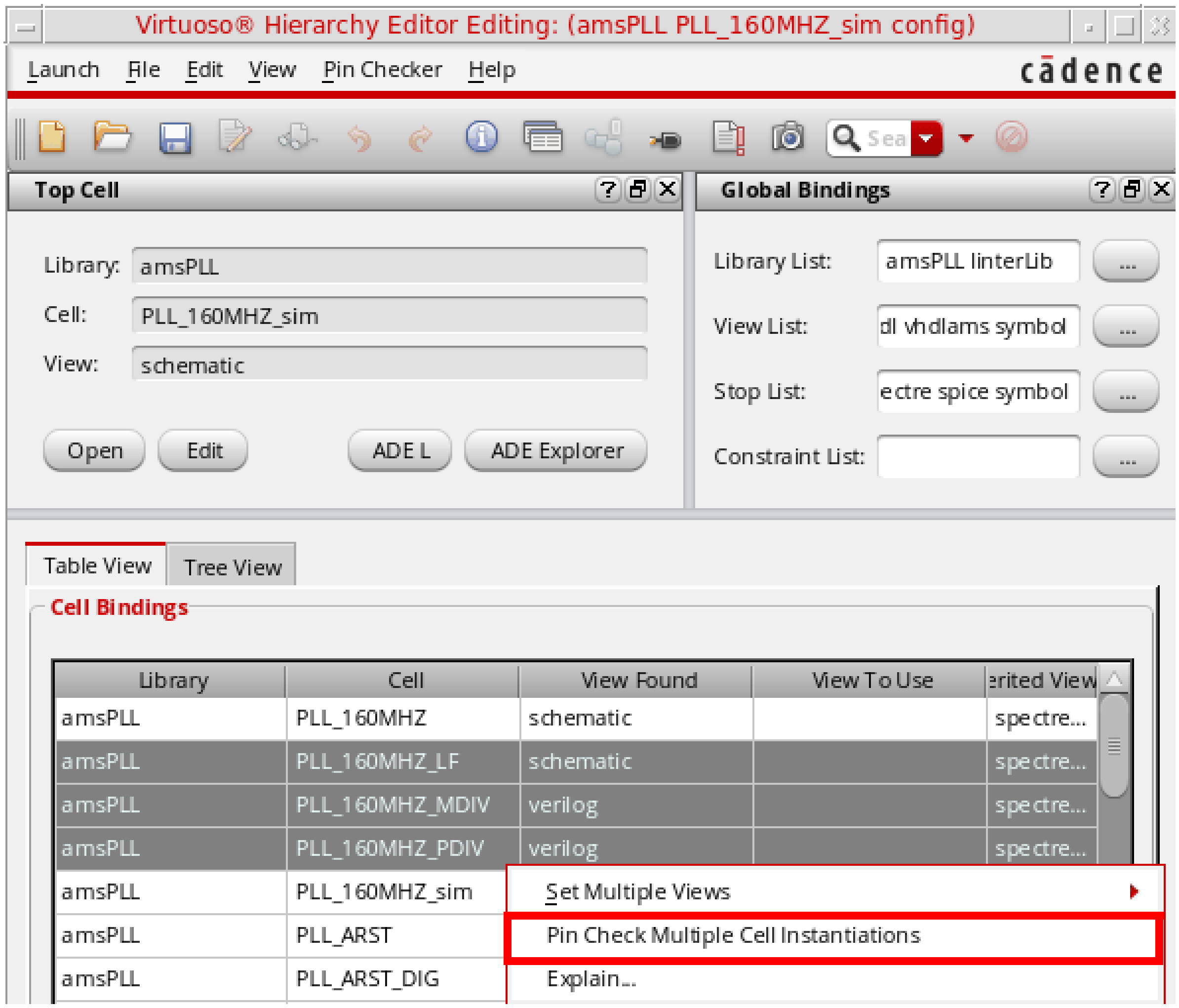
For more information on the Pin Check log file, see Log File Creation.

Multiple Cell Instantiations Pin Check

You can perform pin checking on multiple cells at all the places where they are instantiated in a design. To do this, you need to perform the following steps:

1. In the *Table view* of the Virtuoso Hierarchy Editor window, select multiple cells from the *Cell Bindings* section by using the `Ctrl` or `Shift` key with the left mouse button.

2. Right-click the selected cells and select *Pin Check Multiple Cell Instantiations* from the context-sensitive menu



Library: amsPLL

Cell: PLL_160MHZ_sim

View: schematic

Library List: amsPLL linterLib

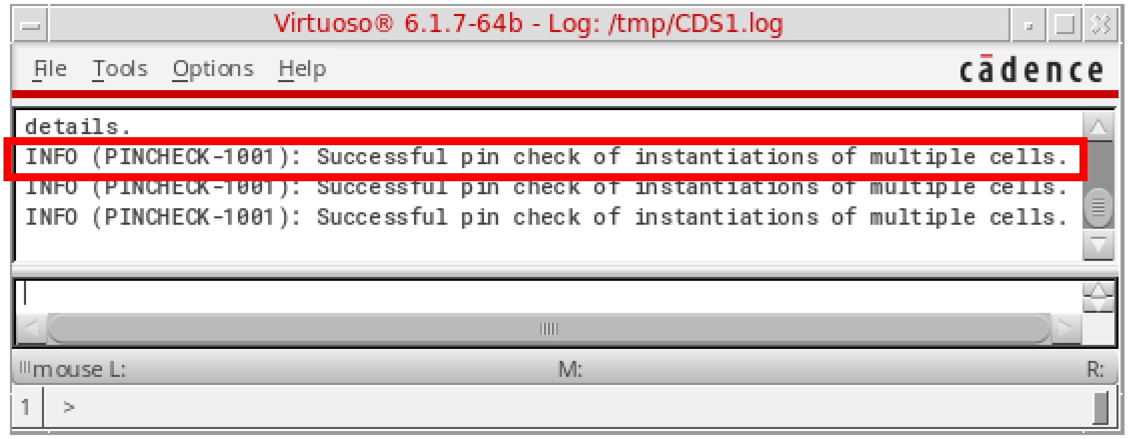
View List: dl vhdlams symbol

Stop List: ectre spice symbol

Constraint List:

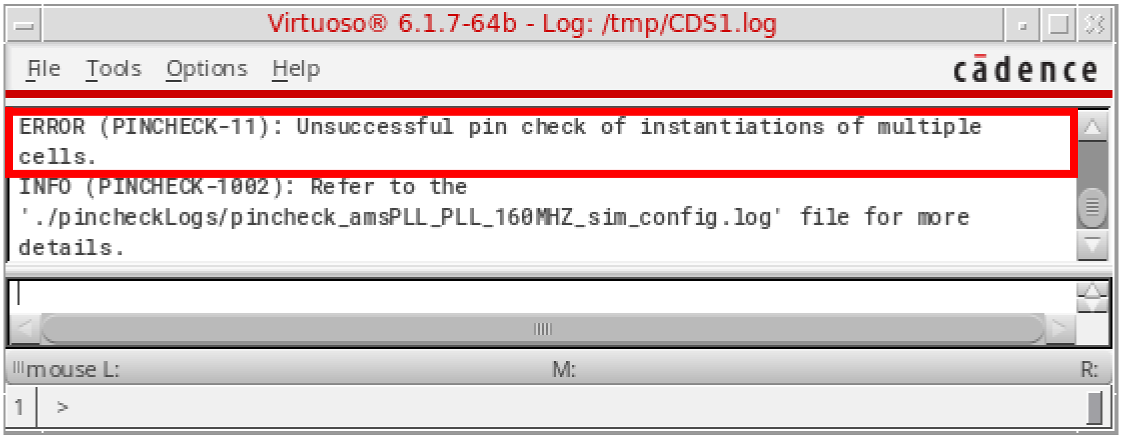
Library	Cell	View Found	View To Use	erited View
amsPLL	PLL_160MHZ	schematic		spectre...
amsPLL	PLL_160MHZ_LF	schematic		spectre...
amsPLL	PLL_160MHZ_MDIV	verilog		spectre...
amsPLL	PLL_160MHZ_PDIV	verilog		spectre...
amsPLL	PLL_160MHZ_sim			
amsPLL	PLL_ARST			
amsPLL	PLL_ARST_DIG			

After Pin Check for instantiations of multiple cells is successfully completed without any error or warning, the successful pin check message is displayed in CIW.



```
details.
INFO (PINCHECK-1001): Successful pin check of instantiations of multiple cells.
INFO (PINCHECK-1001): Successful pin check of instantiations of multiple cells.
INFO (PINCHECK-1001): Successful pin check of instantiations of multiple cells.
```

If there is any error or warning in the design, the unsuccessful pin check message is displayed in CIW and the Pin Check log file is displayed.



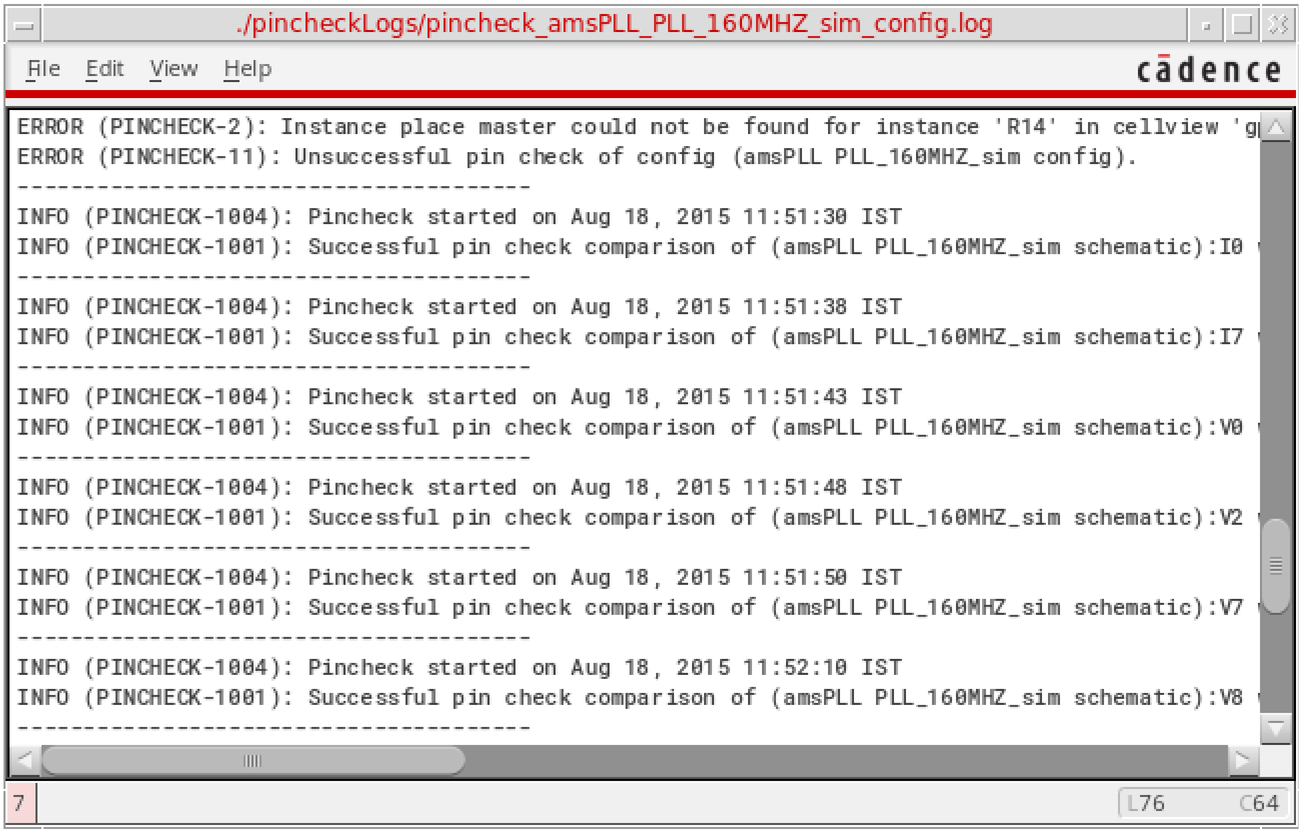
```
ERROR (PINCHECK-11): Unsuccessful pin check of instantiations of multiple
cells.
INFO (PINCHECK-1002): Refer to the
'./pincheckLogs/pincheck_amsPLL_PLL_160MHZ_sim_config.log' file for more
details.
```

For more information on the Pin Check log file, see Log File Creation.

Log File Creation

A log file with the name `pincheck_<library name>_<cell name>_<view name>.log` is created in the current working directory, by default. Here, `<library_name; cell_name; view_name>` are the library, cell, and view names of the current config window where pin checking has been performed. This log file contains the error and warning messages if the pin checker finds any issues during the pin checking operation. For successive pin checking operations, error and warning messages are appended in the log file. A log file window is also displayed showing the content of the log file.

For example, if you are working on a design where the library name is `amsPLL`, the cell name is `PLL_160MHZ_sim`, and the view name is `config`. Then, the Pin Check log file will be created with the name, `pincheck_amsPLL_PLL_160MHZ_sim_config.log`.



```
ERROR (PINCHECK-2): Instance place master could not be found for instance 'R14' in cellview 'g
ERROR (PINCHECK-11): Unsuccessful pin check of config (amsPLL PLL_160MHZ_sim config).
----------------------------------------
INFO (PINCHECK-1004): Pincheck started on Aug 18, 2015 11:51:30 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_sim schematic):I0
----------------------------------------
INFO (PINCHECK-1004): Pincheck started on Aug 18, 2015 11:51:38 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_sim schematic):I7
----------------------------------------
INFO (PINCHECK-1004): Pincheck started on Aug 18, 2015 11:51:43 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_sim schematic):V0
----------------------------------------
INFO (PINCHECK-1004): Pincheck started on Aug 18, 2015 11:51:48 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_sim schematic):V2
----------------------------------------
INFO (PINCHECK-1004): Pincheck started on Aug 18, 2015 11:51:50 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_sim schematic):V7
----------------------------------------
INFO (PINCHECK-1004): Pincheck started on Aug 18, 2015 11:52:10 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_sim schematic):V8
----------------------------------------
```

Changing Log File Creation Path

To change the log file creation path, you need to set the `pcLogFilePath` environment variable in CIW as shown below:

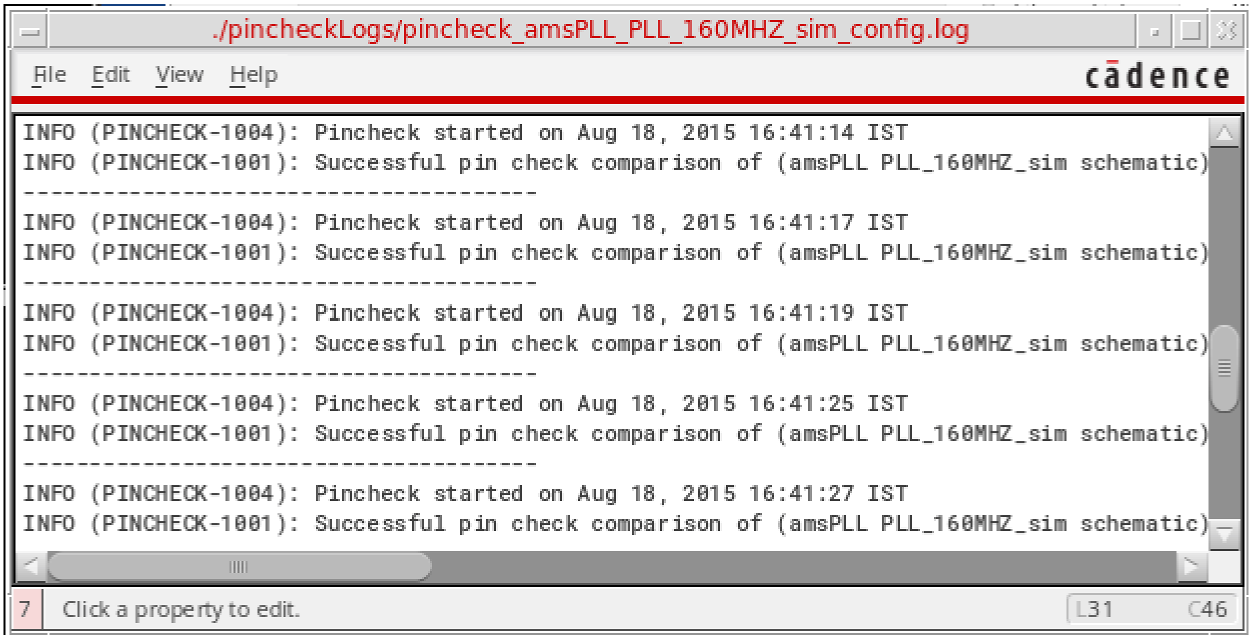
```
envSetVal("pinCheck" "pcLogFilePath" 'string "<file path>")
```

Default value of this environment variable is the current working directory.

Log File Modes

Pin checker logs can be stored in the log file in the `append` or `overwrite` mode:

- `append`: Appends the new pin checking results in the log file as shown below:

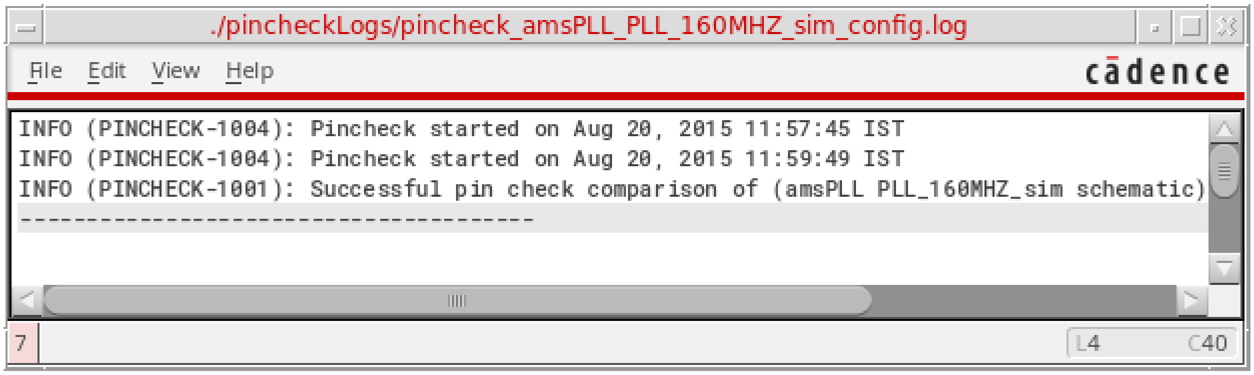

./pincheckLogs/pincheck_amsPLL_PLL_160MHZ_sim_config.log
File Edit View Help
cadence
7
Click a property to edit.
L31
C46

```
INFO (PINCHECK-1004): Pincheck started on Aug 18, 2015 16:41:14 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_sim schematic)
---------------------------------------
INFO (PINCHECK-1004): Pincheck started on Aug 18, 2015 16:41:17 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_sim schematic)
---------------------------------------
INFO (PINCHECK-1004): Pincheck started on Aug 18, 2015 16:41:19 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_sim schematic)
---------------------------------------
INFO (PINCHECK-1004): Pincheck started on Aug 18, 2015 16:41:25 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_sim schematic)
---------------------------------------
INFO (PINCHECK-1004): Pincheck started on Aug 18, 2015 16:41:27 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_sim schematic)
```

To write or store the logs in the `append` mode, you need to set the `pcLogFileOpenMode` environment variable in CIW as shown below:

```
envSetVal("pinCheck" "pcLogFileOpenMode" 'cyclic "append")
```

- `overwrite`: Adds the latest pin checking result in the log file and removes the previous pin checking results from the log file.



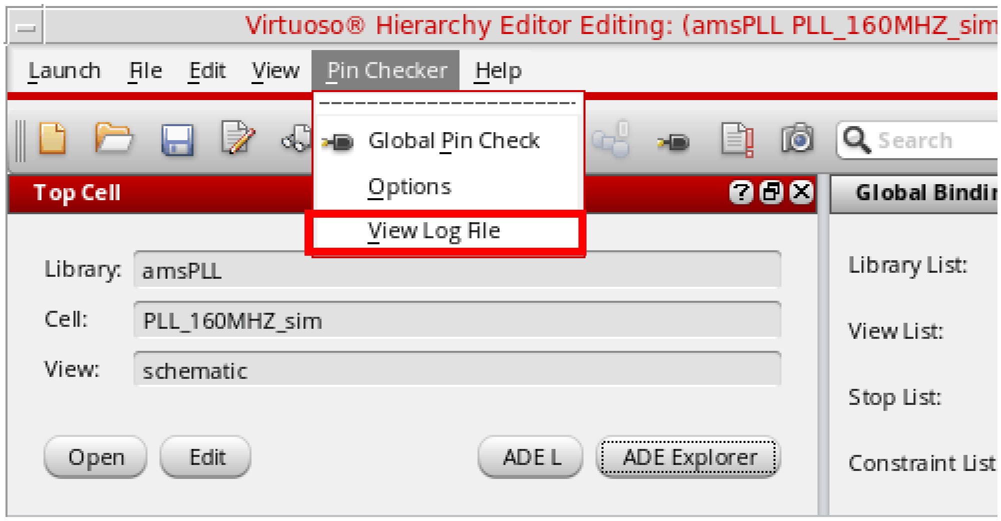
```
INFO (PINCHECK-1004): Pincheck started on Aug 20, 2015 11:57:45 IST
INFO (PINCHECK-1004): Pincheck started on Aug 20, 2015 11:59:49 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_sim schematic)
---------------------------------------
```

To write or store the logs in the `overwrite` mode, you need to set the `pcLogFileOpenMode` environment variable in CIW as shown below:

```
envSetVal("pinCheck" "pcLogFileOpenMode" 'cyclic "overwrite")
```

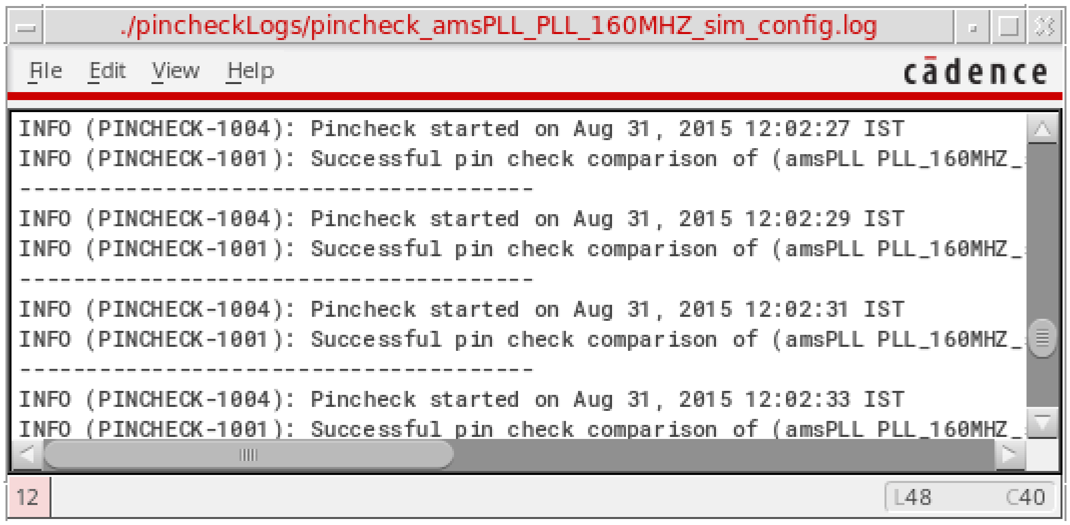
Viewing Log File Window

To view the log file in a window, you need to select *Pin Checker — View Log File*.



Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

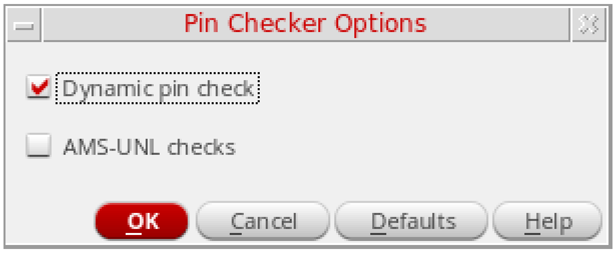
It will show the log file contents in a window.



```
INFO (PINCHECK-1004): Pincheck started on Aug 31, 2015 12:02:27 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_
---------------------------------------
INFO (PINCHECK-1004): Pincheck started on Aug 31, 2015 12:02:29 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_
---------------------------------------
INFO (PINCHECK-1004): Pincheck started on Aug 31, 2015 12:02:31 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_
---------------------------------------
INFO (PINCHECK-1004): Pincheck started on Aug 31, 2015 12:02:33 IST
INFO (PINCHECK-1001): Successful pin check comparison of (amsPLL PLL_160MHZ_
```

Pin Checker Options Form

To open the Pin Checker Options form, select *Pin Checker – Options* from the Virtuoso Hierarchy Editor window.



Pin Checker Options

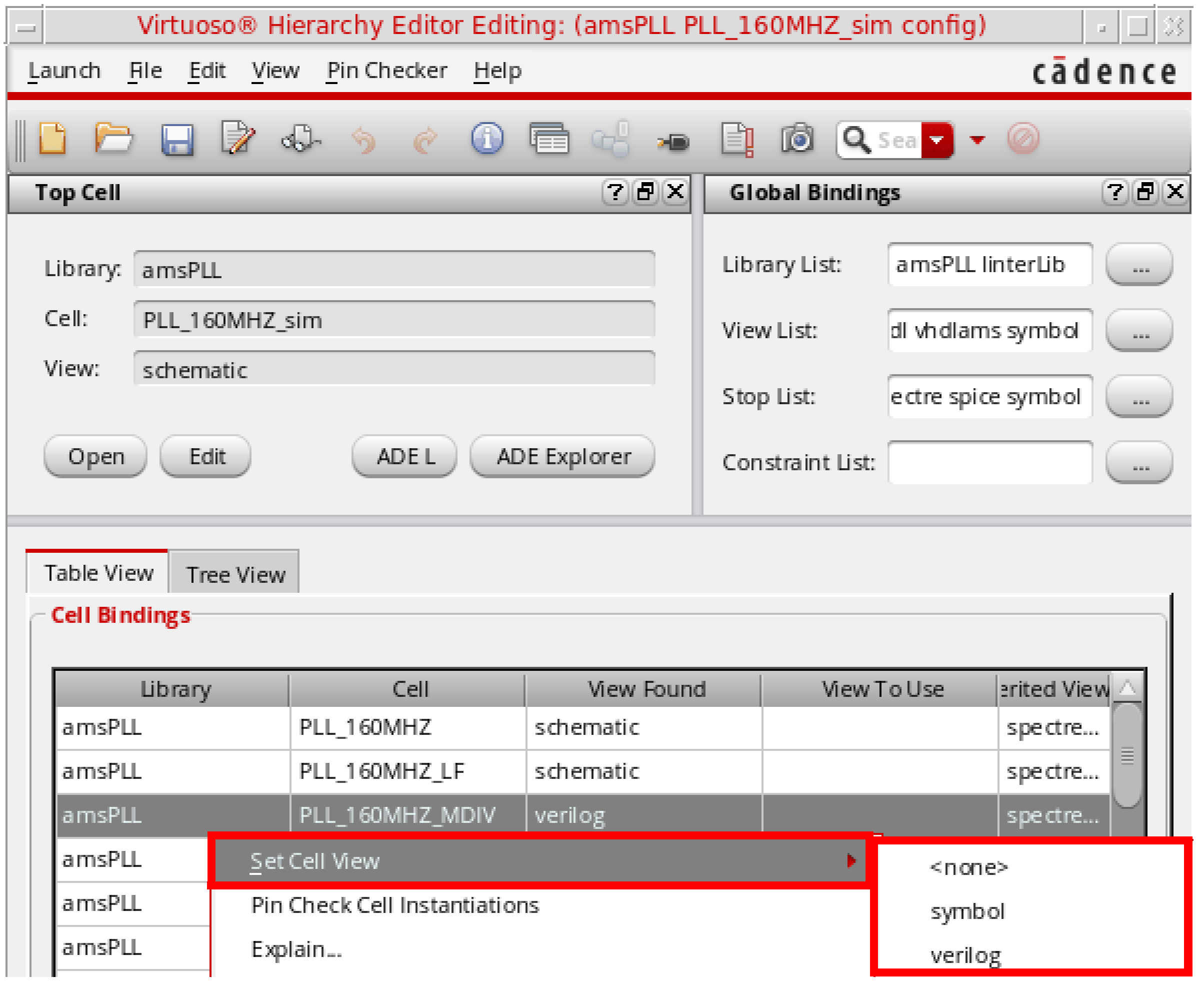
☑ Dynamic pin check

□ AMS-UNL checks

Dynamic pin check

When the *Dynamic pin check* check box is selected, pin checking is automatically performed if the view of a cell or an instance gets changed using the RMB options. The updated value will be applicable to all the open HED windows.

To change the view of a cell, right-click the cell in the *Table View* and select the *Set Cell View* option from the context-sensitive menu.



Top Cell

Library: amsPLL
Cell: PLL_160MHZ_sim
View: schematic

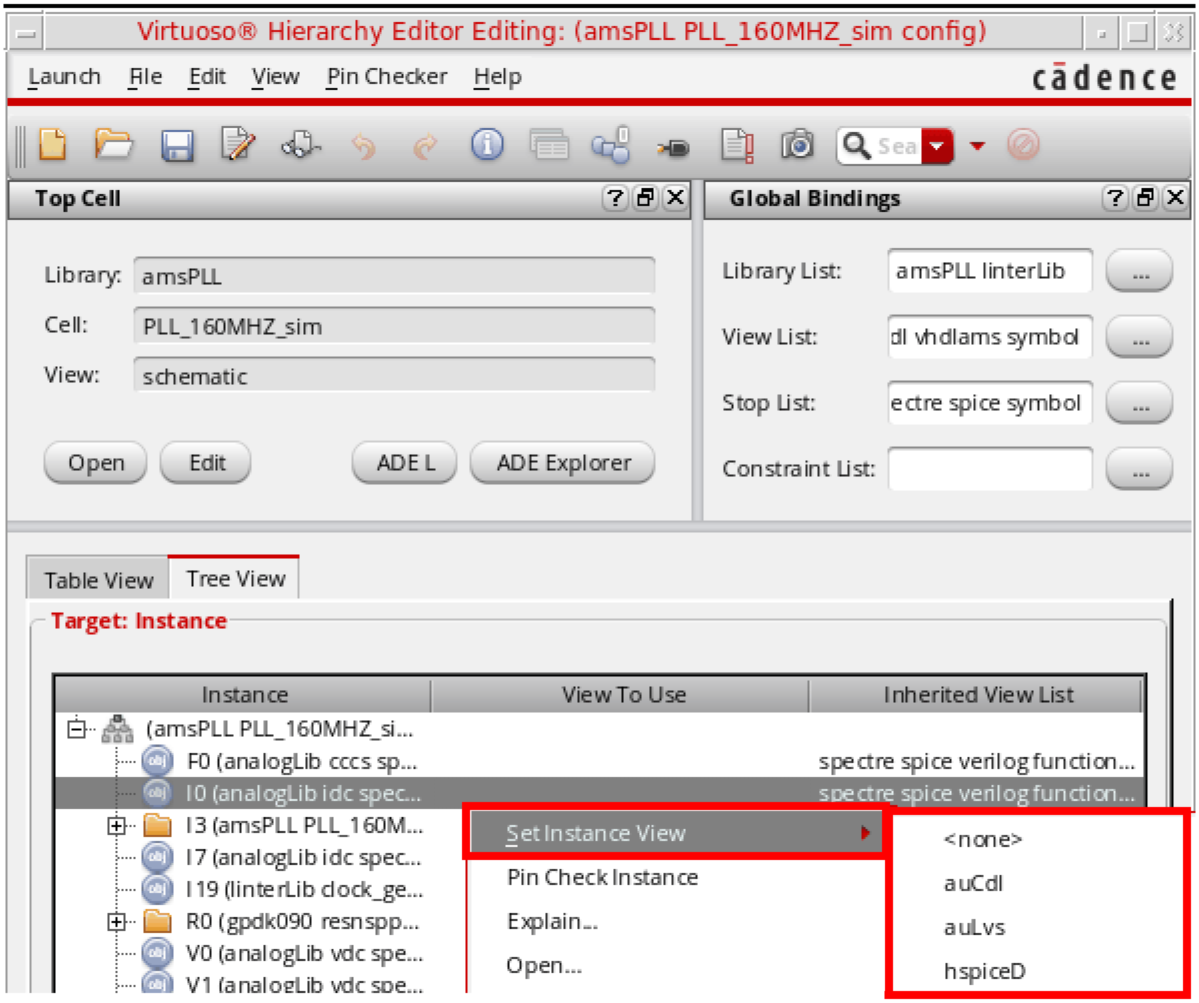
Global Bindings

Library List: amsPLL linterLib
View List: dl vhdlams symbol
Stop List: ectre spice symbol
Constraint List:

Library	Cell	View Found	View To Use	erited View
amsPLL	PLL_160MHZ	schematic		spectre...
amsPLL	PLL_160MHZ_LF	schematic		spectre...
amsPLL	PLL_160MHZ_MDIV	verilog		spectre...
amsPLL				
amsPLL				
amsPLL				

For more information on changing the view of a cell using the RMB option, see Changing View Bindings on a Per-Cell Basis

Similarly, to change the view of an instance, right-click the instance either in the *Table View* or *Tree View* and select the *Set Instance View* option from the context-sensitive menu.



Library:	amsPLL
Cell:	PLL_160MHZ_sim
View:	schematic

Library List:	amsPLL linterLib
View List:	dl vhdlams symbol
Stop List:	ectre spice symbol
Constraint List:	

Instance	View To Use	Inherited View List
(amsPLL PLL_160MHZ_si...		
F0 (analogLib cccs sp...		spectre spice verilog function...
I0 (analogLib idc spec...		spectre spice verilog function...
I3 (amsPLL PLL_160M...		
I7 (analogLib idc spec...		
I19 (interLib clock_ge...		
R0 (gpdk090 resnspp...		
V0 (analogLib vdc spe...		
V1 (analogLib vdc spe...		

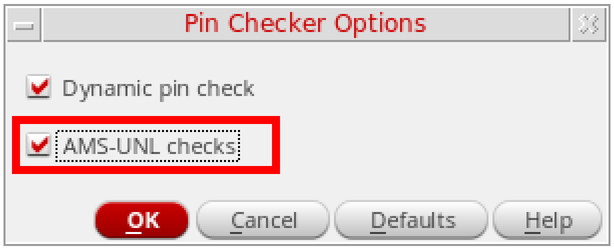
For more information on changing the view of an instance using the RMB option, see Changing View Bindings on a Per-Instance Basis.

By default, the *Dynamic pin check* check box is selected in the Pin Checker Options form.

Note: If the *Dynamic pin check* check box is not selected, you need to explicitly select the *Pin Check Cell Instantiations* or *Pin Check Instance* option from the context-sensitive menu, if the view of that cell or instance is changed.

AMS-UNL checks

When the *AMS-UNL checks* check box is selected, the issues related to the mismatch of vector and split ports on the place master and switch master are resolved in Pin Checker.



Pin Checker Options

☑ Dynamic pin check

☑ AMS-UNL checks

These checks are already implemented in AMS-UNL netlister.

Note: This command will work only if the instance's parent cellview is non-text and the switch master cellview is text.

For example, an instance has the following ports on place master and switch master:

- Place master (symbol) ports: `A<0> B A<1>`
- Switch master (text) ports: `A[0:1] B`

As per the above scenario, the place master has split ports and the switch master has vector ports. Although, both place master and switch master have the same ports, Pin Checker reports an error during the pin check operation because it does the exact matching of the ports.

However, if the *AMS-UNL checks* field is selected, the pin check operation is completed successfully because the Pin Checker compares each member of the vector port of switch master with place master's port.

Note: The place master and switch master can have a mix of vector and split ports. This check is a window specific check. Selecting this field will have effect on that HED window only.

Removing Plug-In Applications

To remove a plug-in application from the Hierarchy Editor,

➤ From the *Plugins* menu, choose the application that you want to remove.

The application menu or toolbar icon disappears from the Hierarchy Editor. Also, in the *Plugins* menu, the application name no longer has a check mark next to it, which indicates that the plug-in is not loaded.

You can load the plug-in application again by selecting it from the *Plugins* menu.

A

The hed.env File

Important

> The Hierarchy Editor now creates and reads the `hed.env` file, instead of the `hierEditor.env` file. The format of the file is different and that only a semicolon can be used as the comment character.

The default display and form settings of the Hierarchy Editor are defined by variables that are registered in the *`your_install_dir`*`/share/cdssetup/hierEditor/env/hed.env` file. You cannot edit this `hed.env` registration file but you can change the settings by creating a local environment file named `hed.env`.

The local `hed.env` file is typically created with the *File – Save Defaults* command and must be in a directory that is listed in your `setup.loc` file (for example, your home directory or current working directory). The local `hed.env` file contains only those settings that are different from the registration `hed.env` file.

The `hed.env` file is read by the Hierarchy Editor when it is started. If you have multiple `hed.env` files, the Hierarchy Editor determines which value to use for each variable based on the search order defined in the `setup.loc` file. For more information about the `setup.loc` file, see, “Cadence Setup Search File: `setup.loc`,” of the Cadence Application Infrastructure User Guide.

Some of the form and display settings, typically the general user interface settings, are saved when you use the *File – Save Defaults* command. For example, the options that you select in the Options form or in the Filters form are saved when you use the *File – Save Defaults* command. Other settings, such as the list of views in the Edit Constants form, can only be changed by editing the `hed.env` file manually.

If you want to set any of the following variables, you must add them to your local `hed.env` file manually:

- `hed.colorChooser bindingError string "`*`color`*`"`

  `bindingError` defines the color used to display data that has a binding error.

Specify the color by name or by RGB value. The name can be any name from the list of SVG color keyword names provided by the World Wide Web Consortium.

The default color is `red`.

- `hed.colorChooser defaultBinding string "`*`color`*`"`

  `defaultBinding` defines the color used to indicate the default binding based on the binding rules.

  Specify the color by name or by RGB value. The name can be any name from the list of SVG color keyword names provided by the World Wide Web Consortium.

  The default color is `black`.

- `hed.colorChooser highlight string "`*`color`*`"`

  `highlight` defines the color used to highlight an object that an external application wants to find. For example, Virtuoso schematic editor users can add an instance probe and see it in the Hierarchy Editor.

  Specify the color by name or by RGB value. The name can be any name from the list of SVG color keyword names provided by the World Wide Web Consortium.

  The default highlight color is `khaki`.

- `hed.colorChooser notInUse string "`*`color`*`"`

  `notInUse` defines the color used to indicate objects that are not in use.

  Specify the color by name or by RGB value. The name can be any name from the list of SVG color keyword names provided by the World Wide Web Consortium.

  The default color is `darkOrange`.

- `hed.colorChooser rowSelection string "`*`color`*`"`

  `rowSelection` defines the color used to indicate a row selection in the table and the tree. Cadence recommends that you do not change the default color, which is set according to Cadence style.

- `hed.colorChooser userBinding string "`*`color`*`"`

  `userBinding` defines the color used to indicate the bindings that you have changed.

  Specify the color by name or by RGB value. The name can be any name from the list of SVG color keyword names provided by the World Wide Web Consortium.

  The default color is `blue`.

- `hed.constants viewChoices string "`*`view1 view2 view3`*` ..."`

`viewChoices` defines the list of views in the *View Choices* list box in the Edit Constants form.

- `hed.display maskLayoutStopLimit int` *`integer_number`* `nil`

  MaskLayout databases larger than the specified integer value (in MBs) will not be opened and will be displayed as stopping in Hierarchy Editor. The default value is `1024`. If *0* is specified, all maskLayout cellviews will be displayed as stopping.

- `hed.display showConstInViewList boolean nil`

  `showConstInViewList`, when set to `nil`, specifies that instead of displaying a constant, the Hierarchy Editor should display the expanded view list that the constant represents. The default value of this variable is `t`.

- `hed.saveAsVerilog filter string "Any Files (*)"`

  This variable defines the filter that is available in the Save As Verilog form. The default filter is `"Any Files (*)"`. To change this to another filter, use the following format:

  [optional comment] (space or semicolon-separated expressions)

  For example:

  ```
  hed.saveAsVerilog filter string "Verilog Files (*.v)"
  ```

- `hed.update autoGenPcdbFiles boolean t`

  `autoGenPcdbFiles`, when set to `t`, specifies that the Hierarchy Editor should automatically generate a `pc.db` file from Verilog or VHDL.

- `hed.update useNCCompilers boolean t`

  `useNCCompilers` is used in conjunction with the `autoGenPcdbFiles` variable. When set to `t`, `useNCCompilers` specifies that the NC compilers (NC-Verilog and NC-VHDL) be used when generating a `pc.db` file from Verilog or VHDL.

Compare Configuration Variables

You can set the following environment variables to control the traversal output displayed while performing the configuration traversal and comparison:

- `hed.configCompare outputDir string "`*`./outputfilepath`*`"`

  Specifies an output directory to save the traversed configuration files. The default output directory is `./configTraversals`.

- `hed.configCompare spaceIndent "`*`spaces`*`"`

Specifies the number of spaces to be used for indentation. The default value is `2` and the value ranges from `1` to `8`.

- `hed.configCompare minTraversal boolean nil`

  `minTraversal`, when set to `t`, performs the minimal traversal of the configuration hierarchy. When set to `nil`, full traversal is performed. The default value is `nil`.

- `hed.configCompare subConfigs boolean nil`

  `subConfigs`, when set to `nil`, specifies that the traversal results of subconfigurations should also be included in the output file. When set to `t`, ignores the subconfiguration traversal results. The default value is `nil`.

- `hed.configCompare diffTool string "`*`tkdiff -w`*`"`

  Specifies the tool to be used to compare the traversal results. The default comparison tool is `tkdiff`.

Comment Characters in the hed.env File

You can use a semicolon to comment out a line in your `hed.env` file. The comment character must be the first character in the line. For example:

```
; My defaults
hed.constants viewChoices string "schematic symbol verilog verilogAMS"
```

B

Customizing Hierarchy Editor Menus

To add custom banner menus or context menu items to a Hierarchy Editor window:

1. Define a UI customization function that will be called whenever a new Hierarchy Editor window is opened and will be passed the `hiWindowId` of the new window.

   The customization function can use hi- SKILL functions to add banner menus to the Hierarchy Editor window or add menu items to the Hierarchy Editor context menus. The customization function can be loaded on Virtuoso startup through the `.cdsinit` file.

2. Register the customization function by using the hedRegUICustomFunc SKILL function. Alternatively, add this call to `.cdsinit` so that the customization function is registered during Virtuoso startup.

   When customizing Hierarchy Editor context menus, the context menus can be accessed using the `cellTableMenu`, `instTableMenu`, and `treeMenu` properties for the Hierarchy Editor window.

   The following properties are also available for the Hierarchy Editor window:

   ❑ `currentItem`: Returns the currently selected item in a Hierarchy editor window.

   ❑ `libName`: Returns the library name of the current configuration.

   ❑ `cellName`: Returns the cell name of the current configuration.

   ❑ `viewName`: Returns the view name of the current configuration.

   ❑ `mode`: Returns the mode that the configuration was opened in. It can be either `r` (representing *read*) or `a` (representing *append*).

   The following sample script defines customization functions that add a custom Hierarchy Editor banner menu and tree context menu item and registers the functions with Hierarchy Editor.

```
procedure(myBannerMenuUIFunc(hedWin)
    myMenu = hiCreateMenuItem(
        ?name 'myMenuItem
        ?itemText "My MenuItem"
        ?itemIcon nil
        ?callback "print(hiGetCurrentWindow()->currentItem)"
        ?disable nil
    )

        hiCreatePulldownMenu(
            'myMenu
            "My Custom menu"
            list(myMenu)
        )

        hiInsertBannerMenu(
            hedWin
            'myMenu
            hiGetNumMenus(hedWin) + 1
        )
    )
)

procedure(myContextMenuUIFunc(hedWin)
    let((treeMI)
        treeMI = hiCreateMenuItem(
        ?name 'treeMI
        ?itemText "My Tree MenuItem"
        ?itemIcon nil
        ?callback "print(hiGetCurrentWindow()->currentItem)"
        ?disable nil
    )
    hiAddMenuItem(hedWin->treeMenu treeMI)
    )
)

hedRegUICustomFunc('myBannerMenuUIFunc)
hedRegUICustomFunc('myContextMenuUIFunc)
```

hedRegUICustomFunc

```
hedRegUICustomFunc(
     s_uifunc
     )
     => t / nil
```

Description

Registers a UI customization function that will be called when a new Hierarchy Editor window is created.

Arguments

s_uifunc	A UI customization function that accepts the `hiWindowId` value of the window being customized.

Value Returned

`t`	The customization function was registered.
`nil`	Registration failed.