cadence®

Verilog In for Virtuoso Design Environment User Guide

Product Version ICADVM20.1
October 2020

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

Contents

Preface . . . 7
Scope . . . 7
Licensing Requirements . . . 8
Related Documentation . . . 8
What's New and KPNS . . . 8
Installation, Environment, and Infrastructure . . . 8
Technology Information . . . 9
Virtuoso Tools . . . 9
Additional Learning Resources . . . 10
Video Library . . . 10
Virtuoso Videos Book . . . 10
Rapid Adoption Kits . . . 10
Help and Support Facilities . . . 11
Customer Support . . . 11
Feedback about Documentation . . . 11
Typographic and Syntax Conventions . . . 13

1
Verilog In . . . 15
Introduction . . . 16
Using Verilog In . . . 18
Prerequisites . . . 18
Memory Requirements . . . 19
Starting Verilog In . . . 19
Importing Data with Verilog In . . . 21
Understanding the Verilog In Graphical User Interface . . . 23
Verilog In Buttons . . . 23
Import Options Tab . . . 24
Global Net Options Tab . . . 32
Schematic Generation Options Tab . . . 34
How Verilog In Imports Data . . . 44

Types of Modules . . . 44
Guidelines for Modifying Your Design . . . 46
Guidelines for Creating and Editing Symbols . . . 47
How Verilog In Imports Modules . . . 51
Using Reference Libraries to Import Incomplete Designs . . . 52
Escaped Name Mapping . . . 53
How Verilog In handles parameters and defparams . . . 53
Data that is not Imported . . . 54
Problems that Might Occur . . . 55
Verilog In Output Files . . . 55
The verilogIn.log File . . . 55
The verilogIn.map.table File . . . 56

2
Verilog In Standalone Mode . . . 57

Introduction . . . 58
Specifying Internal Options and Parameters . . . 58
Starting in Standalone Mode . . . 58
The ihdl_files File . . . 59
Verilog In Options . . . 59
The ihdl_parameter File . . . 61
Parameters . . . 62

3
Verilog 2001 Support . . . 77

Use Model . . . 78
Verilog 2001 Constructs . . . 78
Signed Arithmetic . . . 78
Sized and Typed Parameters and Local Parameters . . . 79
Attributes . . . 80
Inherited Connections . . . 81
Named Parameter Assignment . . . 83
ANSI-C Style Port Declarations . . . 84
Combined Port and Type Declaration . . . 84
Indexed Part Selects . . . 85

Power Operator and Arithmetic Shift Operator 85

A
Pre-Compiled Libraries 87
Introduction 88
Another instance where Pre-Compiled Libraries can be used effectively 89
Creating Pre-Compiled Libraries 89
Using Pre-Compiled Libraries 90
Guidelines for Using Pre-Compiled Libraries 90
Limitations 91
Speeding up the Execution Process 91
Speedup in Case of Netlist View Creation 91
Speedup in Case of Schematic View Creation 91
Speedup Results 92

Preface

Verilog In lets you import a Verilog Hardware Description Language (HDL) file into the Virtuoso design environment. Using this tool, you can create schematic, symbol, and functional views corresponding to the modules in the Verilog file. Verilog In also lets you create netlist views, which are similar to schematics but without the placement and routing information for the design objects.

This user guide describes Verilog In and explains the Verilog 2001 constructs for which Verilog In can create schematics. In addition, the user guide explains how to create and use pre-compiled libraries. This user guide is aimed at the designers of digital 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 Virtuoso Schematic Editor.

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 (for example, ICADVM20.1) releases.

Label	Meaning

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

Licensing Requirements

Verilog In searches for the following licenses in the specified order and checks out one of them:

- 1 license of Virtuoso® Schematic Editor L
- 1 license of Virtuoso® Schematic Editor XL
- 1 license of Virtuoso® Layout Suite L
- 1 license of Virtuoso® Layout Suite XL
- 4 tokens of Virtuoso® Layout Suite GXL

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

Related Documentation

What's New and KPNS

- *Verilog In What's New*
- *Verilog In Known Problems and Solutions*

Installation, Environment, and Infrastructure

- *Cadence Installation Guide*
- *Virtuoso Design Environment User Guide*
- *Virtuoso Design Environment SKILL Reference*
- *Cadence Application Infrastructure User Guide*

Technology Information

- *Virtuoso Technology Data User Guide*
- *Virtuoso Technology Data ASCII Files Reference*
- *Virtuoso Technology Data SKILL Reference*

Virtuoso Tools

IC6.1.8 Only

- *Virtuoso Layout Suite L User Guide*
- *Virtuoso Layout Suite XL User Guide*
- *Virtuoso Layout Suite GXL Reference*

ICADVM20.1 Only

- *Virtuoso Layout Viewer User Guide*
- *Virtuoso Layout Suite XL: Basic Editing User Guide*
- *Virtuoso Layout Suite XL: Connectivity Driven Editing Guide*
- *Virtuoso Layout Suite EXL Reference*
- *Virtuoso Concurrent Layout Editing User Guide*

IC6.1.8 and ICADVM20.1

- *Virtuoso Custom Digital Placer User Guide*
- *Virtuoso Design Rule Driven Editing User Guide*
- *Virtuoso Electrically Aware Design Flow Guide*
- *Virtuoso Floorplanner User Guide*
- *Virtuoso Fluid Guard Ring User Guide*
- *Virtuoso Interactive and Assisted Routing User Guide*
- *Virtuoso Layout Suite SKILL Reference*
- *Virtuoso Module Generator User Guide*

- *Virtuoso Parameterized Cell Reference*
- *Virtuoso Pegasus Interactive User Guide*
- *Virtuoso Space-based Router 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.

In addition, Cadence offers the following training course on schematic views:

- Virtuoso Schematic Editor

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).

For more information, see Getting Help in *Virtuoso Design Environment User Guide.*

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 exactly 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

Verilog In

This chapter describes the following:

- Introduction on page 16
- Using Verilog In on page 18
- Understanding the Verilog In Graphical User Interface on page 23
- How Verilog In Imports Data on page 44
- Verilog In Output Files on page 55

Introduction

When you use Verilog In with Virtuoso® Design Environment, you can convert structural Verilog netlists into one of the following forms:

- Virtuoso schematics
- Netlists in the Virtuoso format
- Verilog text views in a Virtuoso library

In each case, the design is converted into a data format that can be used by Cadence tools. If you convert your Verilog design into schematics, you can edit them in Virtuoso.

The Verilog In software is located in the Cadence software hierarchy at *install_dir*/tools/dfII/bin/ihdl, where *install_dir* is the directory where your Cadence software is installed.

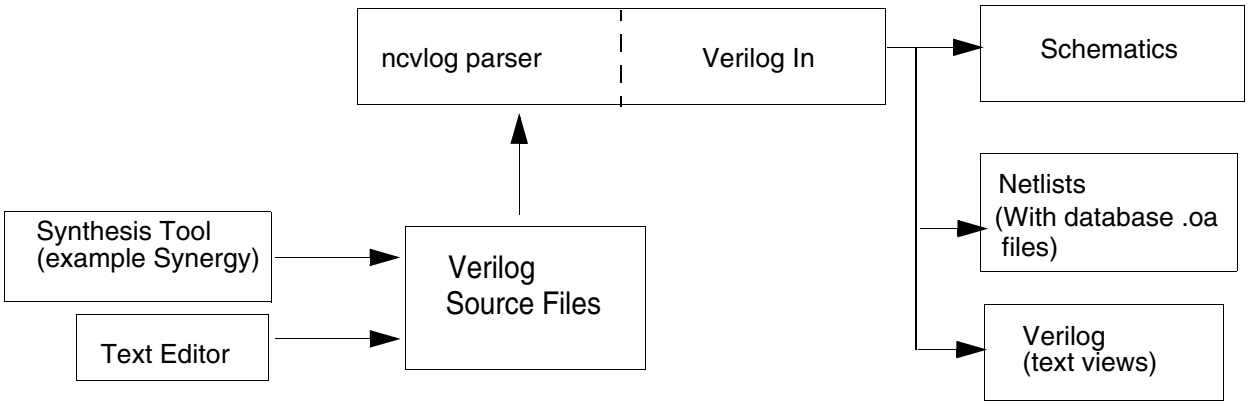
Verilog In uses the ncvlog parser to analyze all designs before they are imported into Verilog.

Important

It is possible that Verilog In is unable to use ncvlog from the appropriate location because the PATH environment variable or LD_LIBRARY_PATH is not set correctly. For details, see "Use Model" on page 78. In this case, Verilog In issues an error. Therefore, ensure that ncvlog is set in the path correctly.

The following figure shows how to use Verilog In with other steps in the Verilog design process.

Figure 1-1 Verilog In Design Flow

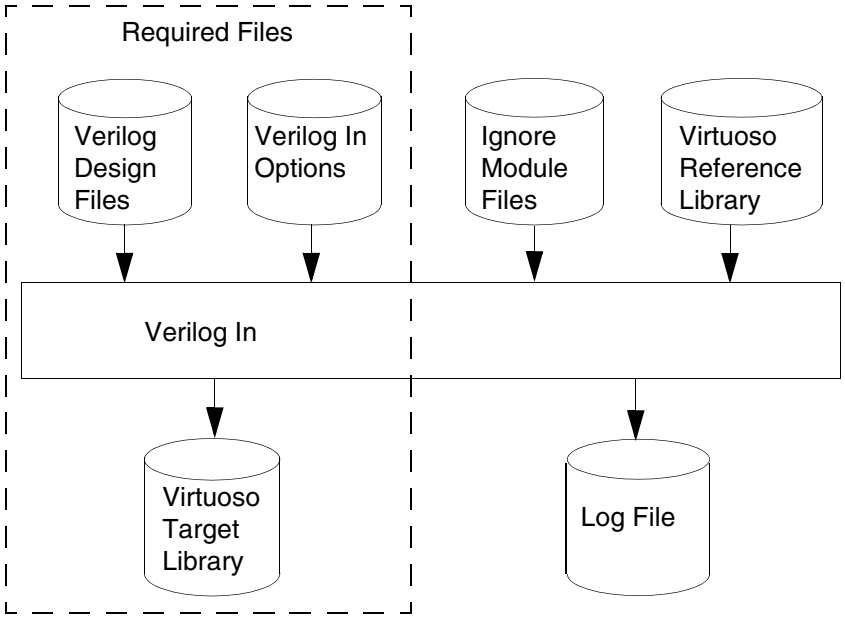


You can use Verilog In to do the following:

- Import a Verilog HDL design into a Virtuoso Design Environment library
- Import a Verilog ASIC library into a Virtuoso Design Environment library
- Import a Verilog HDL design, minus modules that already exist in the library, into a Virtuoso Design Environment library
- Import pieces of a hierarchical design into a Virtuoso Design Environment library
- Import a combination of the above

The following figure shows the files Verilog In uses and generates. The required files are those that you must specify in the Verilog In form.

Figure 1-2 How Verilog In Works



You can run Verilog In by using the Verilog In form as described in this chapter. You access the Verilog In form from the CIW (command interpreter window) in Virtuoso Design Environment.

Chapter 2, “Verilog In Standalone Mode” provides Information about running Verilog In in standalone mode. However, it is recommended that you use the Virtuoso Design Environment user interface to run Verilog In.

Using Verilog In

Verilog In imports a design from a Verilog HDL file into a Virtuoso library.

Prerequisites

Before you use Verilog In, do the following:

- Create the required Verilog design file containing a complete Verilog HDL description file or set of files. (See How Verilog In Imports Data for guidelines about modifying your design files and creating symbols for schematics.)
- Make sure the design has been successfully compiled by the Verilog compiler.

Memory Requirements

The following table lists the memory and swap requirements for your hardware to be able to run Verilog In against different sizes of the input Verilog design. The design size column indicates the amount of instances in the design.

Design size	No. of Modules	Memory Used (Netlist)	Memory Used (Schematic)	Workarea Space Used
210K	141	15.3M	18M	10.7M
341K	3	19.2 M	466M	3.4M
1.8M	328	33.5M	58M	30.3M
2.1M	3	87M	139M	9.2M
3M	9	58M	--	42M
5M	13	56M	--	31M
5M	2265	33M	66M	461M
5.3M	3	61M	258M	22M
26M	2391	151M	236M	243M
83M	383	275M	663M	289M

Starting Verilog In

To start Verilog In

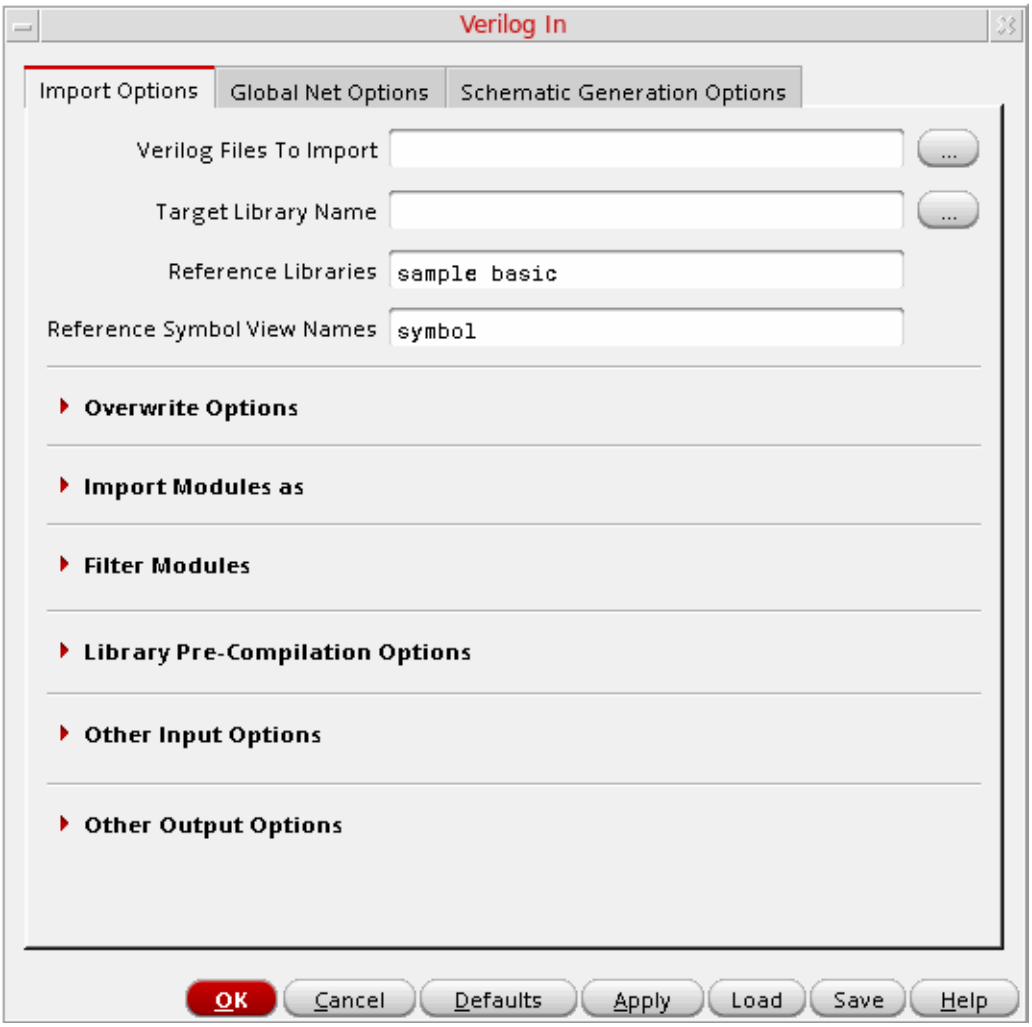
1. In an xterm window, type

   ```
   virtuoso &
   ```

   The Command Interpreter Window (CIW) appears. You use the CIW to control the Cadence software.

2. From the *File* menu, choose *Import — Verilog*.

The Verilog In form appears.

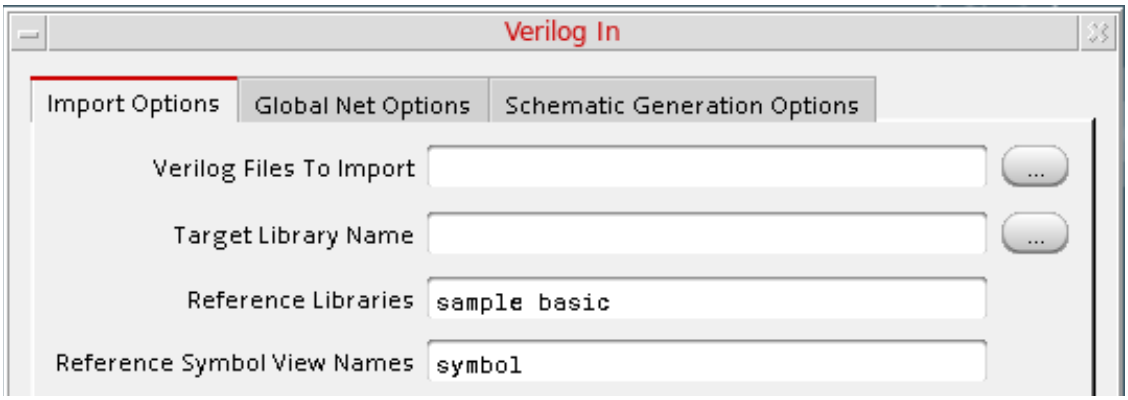


Importing Data with Verilog In

To import data files with Verilog In

1. Type or select the files you want to import in the *Verilog Files To Import* field. You can specify multiple files with paths.
2. Specify the library where the file must be imported in the *Target Library Name* field.
3. Type the names of any reference libraries you need in the *Reference Libraries* field.

   The following figure illustrates an example, where the file `can_counter.v` will be imported to the library `myLib` and will use the reference libraries `sample` and `basic`.



4. Specify any other options you need. For this, expand the relevant section and set the option.

   For example, if you want the imported file to have a custom view name, such as `importedFileSymbol`, instead of the default view name `symbol`, expand the *Import Modules as* section and type the custom name in the *Symbol View Name* field.

   For details on the fields in all tabs, see "Understanding the Verilog In Graphical User Interface" on page 23.

5. Set the global net options, if required.

   a. Click the Global Net Options Tab.

   b. Expand *Global Nets* and specify the global nets, connect nets by name options, and net expressions, as require.

   c. Select and expand *Create Net Expression* to create net expressions and specify the property name for power and ground nets.

6. Set the schematic generation options, if required.

   a. Click the Schematic Generation Options Tab.

   b. Specify the configuration for the schematic to be generated. For this, set the option in the relevant section.

      For example, if you want to generate the schematic without any placement and routing, Deselect the *Full Place and Route* check box.

7. Click *OK* or *Apply*.

Verilog In imports the design into the Virtuoso Design Environment format and places it in the specified library.

Understanding the Verilog In Graphical User Interface

The Verilog In form is organized in three tabs for import options: *Import Options*, *Global Net Options*, and *Schematic Generation Options*. The options available in each tab are categorized into sections. The form always displays the mandatory options, and provides other options within expandable and collapsible sections.

This section describes the tabs and buttons. It includes the following topics:

- Verilog In Buttons
- Import Options Tab
- Global Net Options Tab
- Schematic Generation Options Tab

Verilog In Buttons

The action buttons of Verilog In include the standard buttons: *OK*, *Cancel*, *Defaults*, *Apply*, and *Help*. In addition to these standard buttons, Verilog In includes the buttons *Load* and *Save*, which let you save and reuse parameters (option settings).

By default, the Save Parameters and the Load Parameters forms, which display when you use the *Save* and *Load* buttons respectively, display the `.ihdlParamFile` as the filename. You can use another filename, if required.

Import Options Tab

The Verilog import options are displayed in the first tab, *Import Options*. The options available in this tab are described below.

- Verilog Files To Import
- Target Library Name
- Reference Libraries
- Overwrite Options
- Import Modules as
- Filter Modules
- Library Pre-Compilation Options
- Other Input Options
- Other Output Options

Verilog Files To Import

In this field, type or select the names of the Verilog files that contain the complete descriptions to import. You can type or select multiple files separated by a space.

To select the files, click the corresponding *Browse* button. In the Select File form, browse and select the files you want to import. To select multiple files, hold down the `Ctrl` key and select the files.

Target Library Name

This option lets you specify the Virtuoso Design Environment library in which all the imported cells are to be placed. If the library exists, Verilog In places the imported cells in it. If the library does not exist, Verilog In creates a new target library and places the imported cells in it.

Browse opens the Library Browser form. Clicking on a library name in Library Browser puts that library name in the *Target Library Name* field.

Reference Libraries

Refer to the libraries that contain Virtuoso Design Environment reference cells. Typically, these reference cells are imported as part of another design or are ASIC library cells. If a cell exists in a reference library with the same name as the module to import, Verilog In does not import the module. If you want to generate a multi-sheet schematic, you must specify a reference library containing a sheet border and index sheet symbols (for example, the `US_8ths` library provided by Cadence). The default reference libraries are `sample` and `basic`.

For incomplete designs, if the description of a module is not provided, but a symbol for the module exists in the reference library, it is used to create a structural view for the instantiating module(s). For more details of symbol selection, refer to the section Using Reference Libraries to Import Incomplete Designs.

Reference Symbol View Names

This field lets you refer to multiple symbol view names and find the instance master in the reference library.

Overwrite Options

Set overwrite options for the cellviews to be generated by the import process:

- *Overwrite Existing Views*

  Select this check box if the module to be imported already exists as a cell in the target library and you want to overwrite it with the new version. By default, this check box is not selected and Verilog In does not import a module if it already exists.

  When the *Overwrite Existing Views* check box is selected, Verilog In imports the specified module and overwrites its existing version.

  Note: This option overwrites only the schematic, functional, or netlist views. It does not overwrite the symbol views. To overwrite symbol views, set the *Overwrite Symbol Views* option.

- *Overwrite Symbol Views*

  Set this option if the symbol of the module to be imported already exists in the target library and you want to overwrite it with a new symbol. By default, this option is set to `None` and Verilog In does not overwrite the symbol of a module if it already exists.

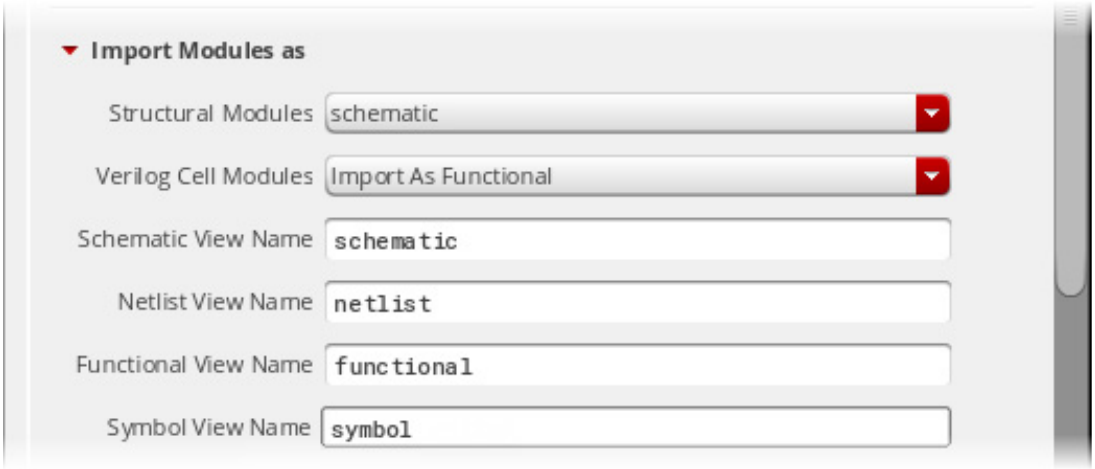
  Set this option to any of the following values to specify which symbols you want to overwrite:

  - *None*: Do not overwrite any existing symbol.
  - *Created by VerilogIn*: Overwrite only those existing symbols that were created by Verilog In. These symbols have the `createdBy` property set as `VerilogIn`. Symbols created by any other tool remain unaffected.

    Note: To view the `createdBy` property of any symbol, open it in *Virtuoso Symbol Editor L*. Choose the *Edit — Properties — Cellview* menu option. The Edit Cellview Properties form that is displayed shows the *createdBy* property field.
  - *Created by TSG and Others*: Overwrite the existing symbols created by the Text-to-Symbol generator or any other tool. For these symbols, either the `createdBy` property is not set or it is set to any value other than `VerilogIn`.

    For more details about the Text-to-Symbol generator, see the Text To Symbol Generator Files section.
  - *All*: Overwrite all the symbols existing in the target library and replace with those imported by Verilog In.

Import Modules as



Set how you want to import the modules using the following options:

- *Structural Modules*

  This option tells Verilog In what views to create for structural modules.

  - `schematic` creates a Virtuoso schematic view.
  - `netlist` creates a Virtuoso netlist view.
  - `functional` creates a Virtuoso functional view.
  - `schematic and functional` creates both a Virtuoso schematic view and a Virtuoso functional view.
  - `netlist and functional` creates both a Virtuoso netlist and a Virtuoso functional view.

  If you use the *schematic* option for a big design containing a large number of gates or huge vector nets, it can take several hours to complete the processing and can require a large amount of memory. In this case, you can create a quick schematic without routed nets and where the connectivity is indicated by names. For generating such a place-only schematic, disable *Full Place and Route* in the *Schematic Generation Options* tabbed form, or specify the Verilog In `+DUMB_SCH` option in the *-f Options* file.
  If you want to maintain the placement and routing information in the schematic, split your Verilog design into sub-designs before you import it using the *schematic* option.
  If you want only the connectivity of the design and not the graphics, use the *netlist* option instead of the *schematic* option.

- *Verilog Cell Modules*

  This option tells Verilog In what to do if a module is described as a Verilog HDL cell in the input file.

  - *Create Symbol Only:* Creates only a symbol view for the Verilog HDL cell.
  - *Import*: Imports the Verilog HDL cell as a view, and creates the database for the view.
  - *Import As Functional:* Imports the Verilog HDL cell as a functional view.

  Note: *`Cell`* refers to Verilog cells as well as to library modules defined in `-v` files or `-y` directories.

- *Schematic View Name*

  Specify the view name for the schematic view to be created in the target library. The default view is `schematic`.

- *Netlist View Name*

Specify the view name for the netlist view to be created in the target library. The default view is `netlist`.

- *Functional View Name*

  Specify the view name for the functional view to be created in the target library. The default view is `functional`.

- *Symbol View Name*

  Specify the view name for the symbol view to be created in the target library. The default view is `symbol`.

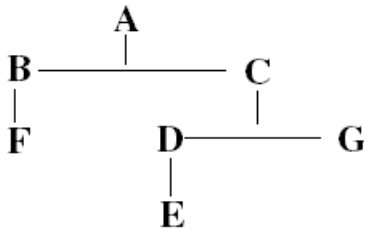
Filter Modules

Set options to filter modules:

- *Ignore Modules File*

  Specify the text file that lists modules you do not want to import. Modules underneath these modules in the design hierarchy are imported, unless they too are listed in this file. You create this file with a text editor and list one module for each line.

  For example, given a module hierarchy as follows:



  If the ignore modules file contains:

```
B
C
```

  Verilog In imports the modules `A`, `F`, `D`, `G`, and `E`.

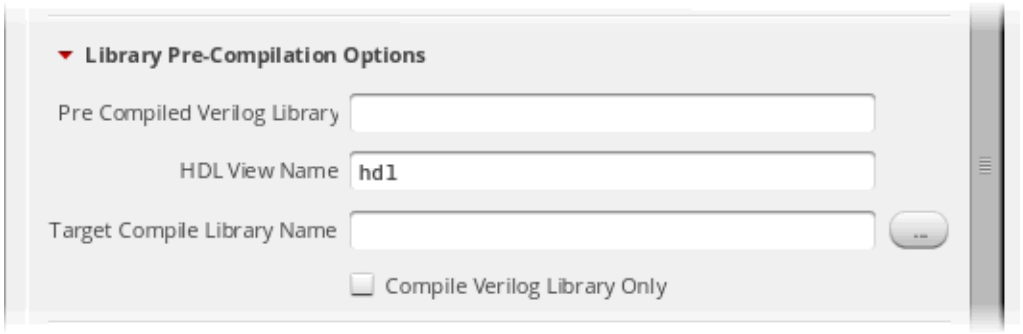
  If a module is listed in the ignore modules file and Verilog In does not find a symbol for the module in the target or reference libraries, Verilog In creates a symbol for the module.

- *Import Modules File*

  Specify the text file, which lists the modules that you need to import. Only the modules listed in the file are imported. The modules underneath these modules in the design hierarchy are marked as `IGNORE` and are not imported. You create this file with a text editor and list one module for each line.

  If the specified text file is empty, all the modules in Verilog In netlists are marked as `IGNORE` and none of the modules is imported. Only the symbol view is created for the modules marked as `IGNORE` while processing the specified text file.

Library Pre-Compilation Options



Specify the pre-compiled libraries:

Note: For more information, see Appendix A, “Pre-Compiled Libraries.”

- *Pre Compiled Verilog Library*

  Specify the name of the pre-compiled library to be used while importing a design. Multiple pre-compiled libraries can be specified by separating them with spaces.

- *HDL View Name*

  Specify the view in the pre-compiled library that is used to find the IR for the cell while using pre-compiled libraries. When pre-compiled libraries are created, use this option to specify the view in which the IR of the module will be saved. The default view name is `hdl`.

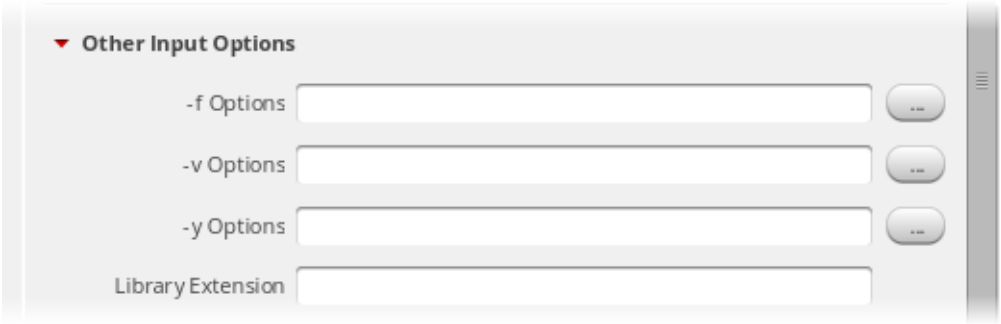
- *Target Compile Library Name*

  Specify the `5.X` library where the Intermediate Representation (IR) produced for the Verilog libraries specified using the `-v` and `-y` options will be located. You can also click the browse button to open the library browser and select a target library.

- *Compile Verilog Library Only*

Select this option if you want Verilog In to only prepare pre-compiled libraries and not import the design.

Other Input Options



Set addition input options:

- *-f Options*

  Specify the names of the files that contain Verilog In options, such as `-y`, `-v`, and other command line options. Verilog In adds these options to the command line options when it imports the file. These option files let you enter the same arguments simultaneously for multiple files. The Verilog In `-f` option files are similar to the *-f option* in Verilog-XL.

- *-v Options*

  Specify the names of the Verilog files needed to compile the design. The reference design files can be given as `-v` options.

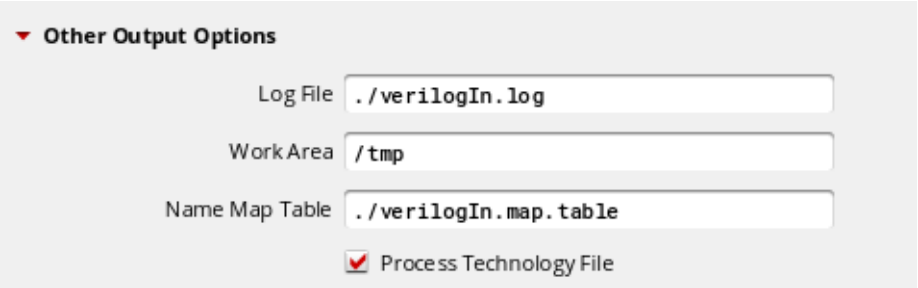
- *-y Options*

  Specify the name of the Verilog library file.

- *Library Extenion*

  Specif a suffix that identifies Verilog files in the path to the Verilog Library. Different organizations use different extensions, such as `.v` (default), `.V`, `.vlog`, or `.verilog`.

Other Output Options



Set additional output options:

- *Log File*

  Specify the file that logs the import status of each module being imported. For more information about the log file, see the "Verilog In Output Files" section. The default log file is `./verilogIn.log`.

- *Work Area*

  Specify the directory in which Verilog In stores internal data. This option is useful for importing large designs when space in the `/tmp` directory is limited. When you quit Verilog In, the software deletes the files in this directory.

- *Name Map Table*

  Specify the name of the map table file that correlates the original names (escaped names that are legal in Verilog but not in Virtuoso Design Environment) to the new mapped names. The default filename is `./verilogIn.map.table` (stored in the current directory).

- *Process Technology File*

  Specify whether you want to create a technology file during import. If you deselect this option, no `tech.db` is created during import. This option can also be specified from the command line using the `-notechfile` switch.

  You can also specify this option using the `processTechFile` environment variable:

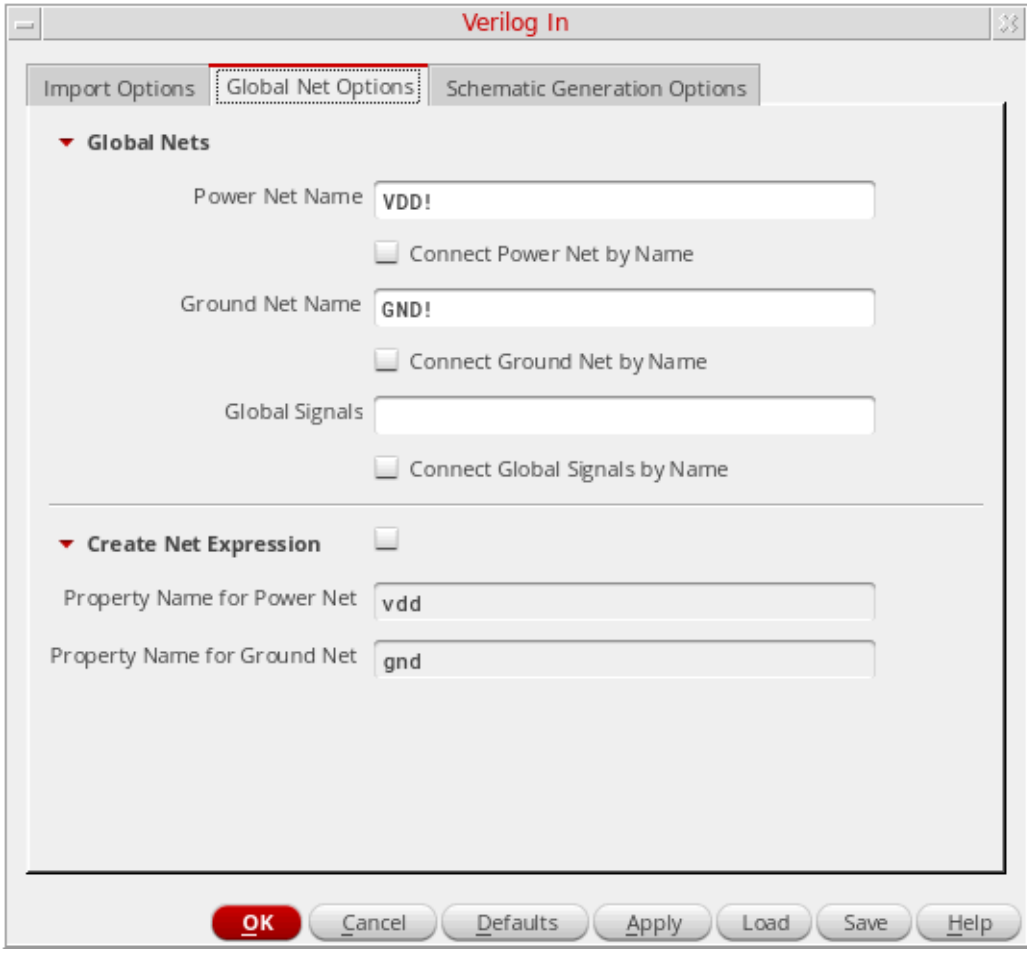
  Type `envSetVal("ihdl" "processTechFile" 'boolean t)` in CIW

  or

  Specify `ihdl processTechFile boolean t/nil` in the .cdsenv file

Global Net Options Tab

The *Global Net Options* tab lets you specify global nets and net expressions of the schematic imported by Verilog In.



Global Nets

Specify names for a global power net, a global ground net, and other global signals in the schematic using the following options:

- *Power Net Name*

  Specify the name for the global power signal In the Verilog design. You can specify only one power net name.

The power net name that you specify is recognized throughout all modules in the design. All `'b1` assignments are replaced by this power net in the schematic. The default name is `VDD!`.

- *Connect Power Net by Name*

  Select this option to connect the power net by the specified power net name.

- *Ground Net Name*

  Specify the name for the global ground signal in the Verilog design. You can specify only one ground net name. The ground net name that you specify is recognized throughout all the modules in the design. All `'b0` assignments are replaced by this power net in the schematic. The default is `GND!`

  - *Connect Ground Net by Name*

    Select this option to connect the ground net by the specified ground net name.

- *Global Signals*

  Specify the names of global signals in the Verilog design other than power and ground. The global signals that you specify are appended in the design with the exclamation character (`!`) to make those signals global. You can specify multiple global signals.

  - *Connect Global Signals by Name*

    Select this option to connect global signals by the specified global signals name.

Important

Use the *Connect Power Net by Name*, *Connect Ground Net by Name*, and *Connect Global Signals by Name* check boxes to import a logical netlist with the power net, ground net, and global signals connected by names, instead of physical wires. Enabling the options to connect by name does not physically connect the nets, but the connections are established by names. In this case, the nets are not routed. One of the cases where you enable these options is when you import CPF where nets can be connected to different power supplies.

Create Net Expression

- *Create Net Expression Check Box*

  Select this option to translate tie highs and tie lows to have net expression created on the power and ground net. When not selected, Verilog connects the port to either a power or a ground signal in case of constant port connectivity.

- *Property Name for Power Net*

  Specify a different net expression property name for power nets. Default is `vdd`.

- *Property Name for Ground Net*

  Specify a different net expression property name for ground nets. Default is `gnd`.

  Note: If you select the *Create Net Expression* option, you cannot leave the *Net Expression Property Name for Power Net* and the *Net Expression Property Name for Ground Net* fields blank.

Schematic Generation Options Tab

The *Schematic Generation Options* tab lets you specify the format of the schematic imported by Verilog In. The options available in this tab are described below.

- Full Place and Route
- Pin Placement
- Text to Symbol Generator Files
- Reference Schematic/Symbol View for Inherited Connections
- Through Cellview to be Used for Port Shorts
- Continuous Assignment Symbol
- Advanced Schematic Generation Options
- Schematic Dimensions

Full Place and Route

Specify placement and routing options for schematics:

- *Full Place and Route*

  Select this check box to generate a schematic with full placement and routing when the number of instances or ports in the file being imported are within the limits specified in

the *Instances Less Than* or *Ports Less Than* fields. If the number of instances or ports exceed the specified number, Verilog In generates a place-only schematic in which the nets are not routed and the connectivity is indicated by name.

Deselect this check box to create a place-only schematic, irrespective of the number of instances or ports. When you deselect the check box, the *Instances Less Than* and *Ports Less Than* fields become disabled.

Notes:

- If you select *Full Place and Route* and the design being imported has a large number of instances and ports, the schematic generation process can take significant time to place and route the instances and nets.
- To always create a place-only schematic, you can specify the Verilog In `+DUMB_SCH` option in the *-f Options* file or deselect the *Full Place and Route* check box.
- Place-only schematics are recommended for importing large benchmarking designs.

- *Instances Less Than*

  Specify the maximum number of instances in the file being imported when the *Full Place and Route* check box is selected. If the file has more than the specified number of instances, Verilog In generates a place-only schematic.

- *Ports Less Than*

  Specify the maximum number of ports in the file being imported when the *Full Place and Route* check box is selected. If the file has more than the specified number of ports, Verilog In generates a place-only schematic.

The following table specifies the default value and the environment variable of these options, which are typically set in `.cdnsenv`.

Field	Default Value	Environment Variable
Instances Less Than	`20000`	`pnrMaxInst`
Ports Less Than	`5000`	`pnrMaxPort`

For information on the corresponding parameters for the `ihdl_parameter` file, see Chapter 2, “Verilog In Standalone Mode.”

Pin Placement

Specify the direction for placing pins in the schematic that Verilog In will generate using the pin placement options:

- *Pin Placement Configuration*
  - *Left and Right Sides*

    Place all pins on the left and right sides of the symbols, usually with input pins on the left, and inout pins and output pins on the right.
  - *All Sides*

    Place the pins on any side.
  - *Pin Placement File* and *Pin Placement File Name*

    Specify a pin-placement filename in the *Pin Placement File Name* field. In this file, you can specify the direction of each pin in the imported module.

    The syntax of the pin-direction declaration in a pin-placement file includes the module name, the pin direction, and the pin name list. The values for the direction are `top`, `bottom`, `left`, or `right`. The pin names are specified as a list of strings separated by spaces and are the existent pin names on the given module name. You can repeat this statement many times, but do not duplicate the same module-pin pair.

    The syntax of entries in a pin placement file is as follows:

    ```
    pin placement := module direction pinNames
    ```

    Note: The pin names in the pin-placement file must match the names specified in the port list. Additionally, the pin names must not contain spaces, except escaped names.

    The following is an example entry in a pin placement file:

    ```
    pin_placement := test_module top B C D[0] {A,B,A[2:3],D,E}
    ```

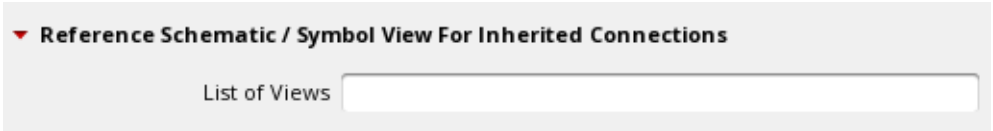
For compatibility with previous product versions, the pin-placement functionality also supports a comma-separated pin-placement file. The following is an example entry in a comma-separated file:

```
pin_placement := test4, top, a, b
```

The following table describes the behavior of the pin-placement functionality, based on the format of the entries in the pin-placement file.

Format	Behavior
A space-separated file containing scalars, vectors, and bundles in the correct syntax	Places the pins as expected.
A comma-separated file containing only scalars	Places the pins as expected.
An incorrect comma-separated file that contains vectors and bundles, or incorrect syntax	Does not place the pins as expected. Only the import process is completed.
A space-separated file where the syntax is incorrect	Causes an error and exits the import process.

Reference Schematic/Symbol View for Inherited Connections



Specify space-separated schematic or symbol views in the *List of Views* field, which the tool can look up for missing terminals on an instance line in the input Verilog file.

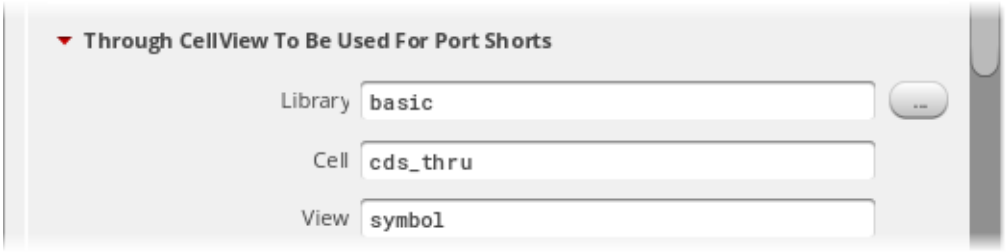
It is possible that an instance line in the Verilog file has more terminals than the number of terminals specified on the identified instance symbol master. In this case, the tool uses the schematic or symbol views specified in the *Reference Schematic/Symbol View For Inherited Connections* field to look for the required additional terminals. The reference schematic views can contain terminals or nets with net expressions whose names match the names of the additional terminals on the instance line. The tool extracts the `netExpression` property name from those inherited terminals or nets. It then sets the `netSet` property on the instance for the missing terminals, as described below.

- The `netSet` property name is set using the `netExpression` property name obtained from the reference schematic

- The `netSet` property value is set from the net connected to the instance line in the Verilog file

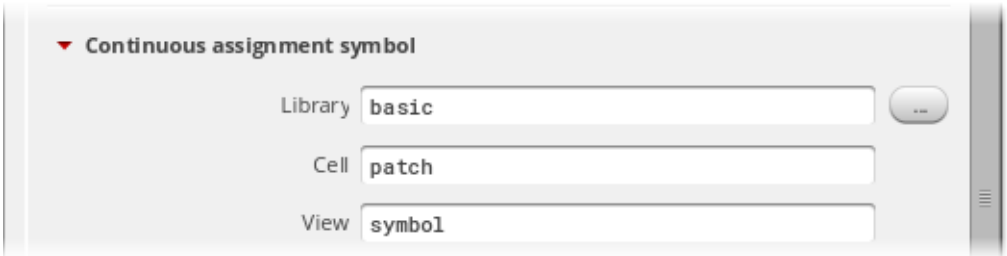
For example, you can use this field when you want to import a physical Verilog netlist with power-ground terminals on an instance line, where the corresponding symbol master does not have the power-ground terminals. In this case, Verilog In checks if the power-ground terminals or nets exist in the reference schematic/symbol view list. If the corresponding terminals or nets exist and are inherited, the application creates the `netSet` properties for these power-ground terminals on the instances in the schematic.

Through Cellview to be Used for Port Shorts



In their corresponding input fields, specify the library, cell, and view name of the component to be used between shorted ports. When the input and output ports of a module in the input Verilog design are shorted, Verilog In puts the default symbol called `cds_thru` between the shorted ports. The symbol `cds_thru` is put instead of the patch symbol used for other shorts to avoid shorted terminals connectivity extraction errors from Virtuoso Schematic Editor. The default location of this symbol is `basic`, `cds_thru` and `symbol` for library, cell and view name, respectively. You can customize the location of the symbol from the Schematic Generation Option form. You can also click the browse button and select the design from Library Browser.

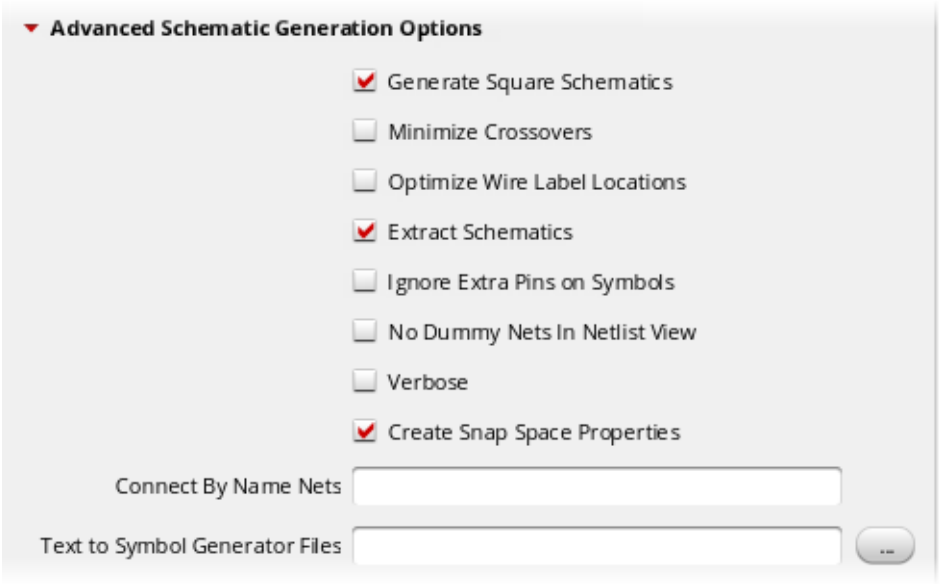
Continuous Assignment Symbol



In their corresponding input fields, specify the library, cell, and view names for the patch symbol and avoid net shorting. Net shorting happens whenever you use an assign statement,

such as `assign a=b;` in which both `a` and `b` are local nets. You can also click the browse button and select the design from Library Browser.

Advanced Schematic Generation Options



Set advanced schematic generation options:

- *Generate Square Schematics*

  Specify whether to square the schematic. Turn this option *off* if you do not want to have Verilog In manipulate rows and columns of devices to make a rectangular schematic into a square one.

  The default is `on` (enabled).

- *Minimize Crossovers*

  Specify whether to minimize crossover of nets. Turn this option `on` to minimize crossovers of nets.

  The default is `off` (disabled).

- *Optimize Wire Label Locations*

  Specify whether to override default label placement. Turn this option *on* to override default label placement, which keeps overlap of segments or labels to a minimum, in favor of fast placement, which places labels of segments at the midpoint without checking for the minimum overlap.

The default is `off` (disabled).

- *Extract Schematics*

  Specify whether to extract the schematic. Turn this option `on` to extract the schematic, that is, to have Verilog In look for errors and warnings in the schematic written into the Open Access database format. An extracted schematic shows blinking markers for the errors or warnings in the schematic view.

  The default is `on` (enabled).

- *Ignore Extra Pins on Symbol*

  Control the selection of symbols from the reference libraries. If this button is *on* and Verilog In finds a reference symbol with the same name as specified in the Verilog design, the symbol will be picked up. The pins not referred will remain unconnected in the schematic.

  The default is `off` (disabled).

  Note: The symbol will be picked up even if all its pins are not used.

  To get the pin symbols from the reference library, set the following environment variables:

  ```
  ihdl    pin_master_cells    string    "ipin opin iopin"
  ```

  The above variable specifies the cell names to be used for IO pins. The sequence for specifying the pin names is input, output, and input-output pins. The default value is `"ipin opin iopin"`.

  ```
  ihdl    pin_master_basic_lib    boolean    nil
  ```

  The above variable specifies whether the symbol views of pins should be taken from the basic library or from the reference libraries.

  The default value is `t`. When set to `t`, the offsheet symbol views of pins are taken from the basic library and the symbol views of pins are taken from the reference libraries. If any of the symbol views are not found in the respective libraries, then the symbol views are created in the destination library.

  When set to `nil`, all symbol views of pins are taken from the reference libraries. The cell names for pins are determined by the `pin_master_cells` variable.

- *No Dummy Nets In Netlist View*

  Specify whether to connect dummy nets to unconnected pins of instances. Deselect this option (`off`) to connect dummy nets named `_NeTt_1,2,3...` in the netlist view to unconnected pins of instances.

The default is `off` (disabled).

Note: This option only works for netlist view and not for schematic view. For schematic view all unconnected pins on instances will always get connected to dummy nets named `NeTt_1,2,3....`

- *Verbose*

  Select to print detailed status messages while the schematic is being partitioned and routed. Turn this option *on* to print detailed messages.

  The default is `off` (disabled).

- *Create Snap Space Properties*

  Enable or disable the creation of snap space properties. Snap space can be defined as the minimum distance the cursor moves, in user units. The default distance (0.0625) is equal to half the default grid spacing distance. When this option is set to 0, then the x-y Snap Space properties are not created on the cellview.

  The default is `1` (enabled).

- *Connect By Name Nets*

  Type the list of scalar nets that need to be connected by name instead of physical wires. If a net specified in this field is a part of a bundle then all the nets in the bundle are connected by name.

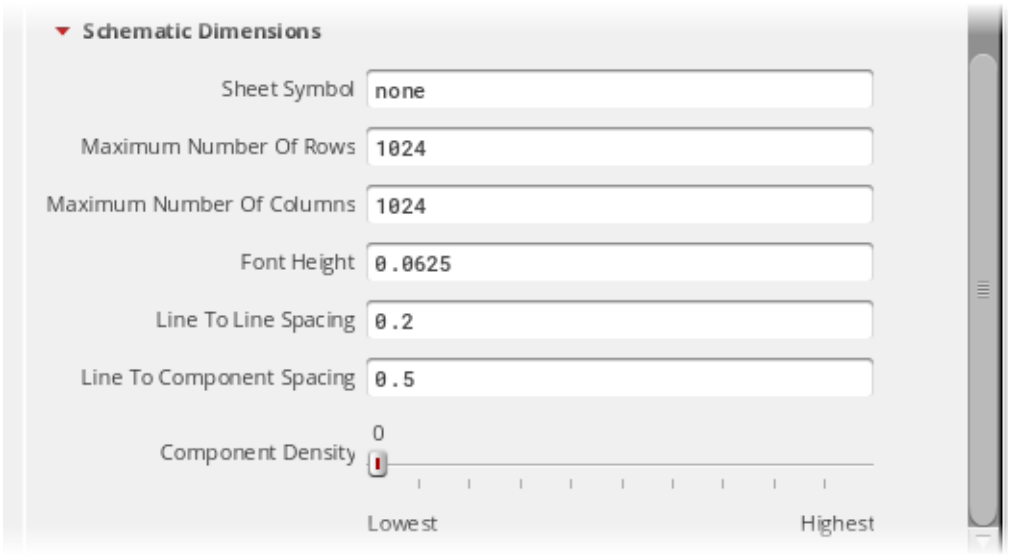
- *Text to Symbol Generator Files*

  Specify a space-separated list of `tsg` files to be used by Verilog In to generate symbols in the target library. Verilog In internally runs the Text-to-Symbol generator, a tool that reads the symbol descriptions given in the `tsg` files to create symbol views.

  For more details about the Text-to-Symbol generator and the `tsg` files, see Text-to-Symbol Generator in Virtuoso Schematic Editor L User Guide.

If the `tsg` files are used, the direction of pins specified by the Pin Placement option is ignored. Pins are placed on symbols as defined in the `tsg` files.

Schematic Dimensions



Set schematic dimenations:

- *Sheet Symbol*

  Specify the symbol that controls the size and page orientation of the sheet on which the schematic is to be made. Valid entries are the names of customer-designed sheet symbols and sheet symbols offered by Virtuoso Design Environment.

  A selection of sample sheet symbols is shipped with Virtuoso Design Environment in the *install_dir*`/tools/dfII/etc/cdslib/sheets/US_8ths` library. These symbols include metric sheet symbols `A0` through `A4` and sheet symbols for the traditional `A`, `A.book` (vertical orientation), `B`, `C`, `D`, `E`, and `F` sizes.

  The library containing the sheet symbol entered in this field must be in your `cds.lib` file so that Verilog In can find the correct information.

  If the *Sheet Symbol* is `none`, Verilog In makes a single sheet schematic. If a specific sheet symbol size is `provided`, and Verilog In cannot fit the schematic onto one sheet, it creates a multisheet schematic. A multisheet schematic has one index sheet and many schematic sheets.

  The view with the index sheet for the schematic has the same name as the Verilog module. The schematic sheets are numbered `cellname.sheetnnn` where `cellname` is the Verilog cell name and `nnn` is the sheet number.

  The default is `none`.

For information about working with sheet symbols and multi-sheet schematics, refer to *Virtuoso Schematic Editor User Guide*.

- *Maximum Number Of Rows*

  Specify the maximum number of rows that Verilog In places on each sheet. Verilog In uses this option only on a multi-sheet schematic. The value must be an integer from 1 to 1024.

  Depending on the size of the instance symbols and the size of the sheet, Verilog In might not be able to place the maximum number of rows on a sheet. If you specify a value that is too large for the size of the instance symbols and the size of the sheet, Verilog In places as many rows as will fit on each sheet up to the maximum of 1024.

  The default is `1024`.

- *Maximum Number Of Columns*

  Specify the maximum number of columns that Verilog In places on each sheet. Verilog In uses this option only on a multi-sheet schematic. The value must be an integer from 1 to 1024.

  Depending on the size of the instance symbols and the size of the sheet, Verilog In might not be able to place the maximum number of columns on a sheet. If you specify a value that is too large for the size of the instance symbols and the size of the sheet, Verilog In places as many columns as will fit on each sheet up to the maximum of 1024.

  The default is `1024`.

- *Font Height*

  Control the size of the font used for pin, wire, and instance labels. The value must be a real value between 0.0375 and 0.125. The wire and instance labels use the font size specified in *Font Height*. Pin labels are scaled down to 75 percent of the specified size.

  The default is `0.0625`.

- *Line To Line Spacing*

  Specify the spacing in inches between nets flowing in a channel for each sheet. The spacing for net segments connected to instance pins depends on the pins placed on symbols of the instances. Line To Line Spacing is used for all other net segments. The value must be a decimal number in the range of 0.19 to .5 inches.

  The default is `0.2`.

  Note: The line to line spacing range is 0.125 to 0.625 inches.

- *Line To Component Spacing*

  Specify the spacing in inches between a component and the nearest net flowing in a channel. The value must be a decimal number in the range of 0.19 to .5 inches.

  The default is `0.2`.

- *Component Density*

  Control the density of the schematic. The value must be an integer from 0 to 100, where 100 is the most dense and 0 is the least dense.

  The default is `0`.

How Verilog In Imports Data

This section describes

- How Verilog In classifies modules
- Guidelines for modifying your design
- Guidelines for creating schematic symbols
- How Verilog In imports modules
- Escaped name mapping
- How Verilog In handles parameters and defparams
- Why some data is not imported
- Problems that might occur

Types of Modules

Verilog HDL format is a textual description of the design. This format describes each design as a collection of modules. For each module in the design, Verilog In creates a cell in the Virtuoso Design Environment library. Verilog In looks at each module and decides if it is

- A behavioral cell module, which is a module defined as a set of procedures
- A structural cell module, which is an interconnection of instances
- A Verilog HDL cell module

Verilog In treats User-Defined Primitives (UDPs) like modules.

Regardless of the type of module, the imported design retains its connectivity between module instances.

Behavioral Cell Modules

If the module is behavioral, Verilog In imports it into a functional view and a symbol view. Verilog In decides that a module is behavioral if importing the module as a schematic does not capture all the information in the HDL description. In this case, the netlister might not be able to regenerate the HDL description from the schematic view of the module. Accordingly, Verilog In considers a module to be behavioral if any of the following is true:

- The module has a specify block statement
- The module has an instance of the `cds_alias` module
  (Verilog In treats the `cds_alias` module as a special case.)
- The module has global or scoped nets
- The module has a child with multiple ports with the same name
- The module has an expression on a port
- The module has strengths defined for a gate
- The module has a task call, function call, or declaration
- The module has a delay statement
- The module has any behavioral statement
- The module has a reg declaration
- The module has compiler directives. Note the exceptions described below:

  A module is not always considered behavioral only because it has compiler directives. When a schematic is generated for a module that has compiler directives, all the compiler directives are ignored except `` `timescale ``, which is added as a property on the schematic, `` `define ``, and `` `ifdef. ``

  The `` `include `` and `` `define `` compiler directives are not always considered behavioral. Refer Guidelines for Modifying Your Design.

Structural Cell Modules

If the module is structural, Verilog In imports it into a schematic view and a symbol view. Verilog In decides a module is structural if it has only the following types of statements:

- Module or UDP instantiations
- Gate instantiations
- Input and output ports, and wire net declarations

If the module has behavioral statements or is classified as behavioral (see the previous section), it is imported as a functional view.

Verilog HDL Cell Modules

If the module is a Verilog HDL cell, Verilog In uses the *Verilog Cell Modules* option on the Verilog In form to import the module as a functional view and create a symbol for it. If insufficient information is provided in the module to create a functional view, Verilog In creates only the symbol.

Verilog In decides a module is a Verilog HDL cell if it meets one of the following conditions:

- The module definition is in a `-y` directory or `-v` file, and the Verilog option `+nolibcell` is not specified.
- A `'celldefine` compiler directive precedes the module.

For a description of a Verilog HDL cell, see the *Verilog-XL Reference*.

Guidelines for Modifying Your Design

The following section describes how Verilog In imports data and tells you how to modify the design before and after you import it.

- You must modify Verilog descriptions that are order-dependent with respect to compiler directives (for example, `'define`) to remove this order dependency before they are imported (or entered using Virtuoso Schematic Editor).

  All compiler directives necessary for the correct interpretation of a module must be contained in the textual cellview describing that module in the Virtuoso Design Environment database. You can use the include file facility to include a set of `'define` statements from a single file with all modules that use the resulting text macros.

- Verilog-XL `'include` files are not managed in Virtuoso Design Environment. You must manage included files in your own directories.

  You must specify the full path names for included files or other required files (for example, memory files) in Verilog descriptions.

- Verilog In inserts files that are included by the `'include` directive into the module.

Verilog In creates a schematic if possible. Otherwise Verilog In creates a functional view, which does not display the included file.

- An imported schematic that instantiates a lower level module uses an improper symbol for the lower level module when
  - A symbol exists for the lower level module before the import begins
  - A mismatch exists between the ports on the lower level module that is imported and the existing symbol of the cell with which the lower level module is associated

  You must either delete the existing symbols from your design file before you import the lower-level module or turn on the *Overwrite Existing Views* option on the Verilog In form.
- Verilog In might create incorrect schematics under certain conditions. In these cases, Verilog In reports in the log file that a problem occurred while creating or checking connectivity of the schematic.

  You must edit and correct the schematic.
- Verilog In saves the following information as properties on schematic views:
  - The `'timescale` directive
  - The `rise and fall` delay on Verilog primitives
  - The `tri`, `tri0`, `tri1`, `trireg`, `wand`, `wor`, `triand`, `trior`, `supply0`, and `supply1` net declarations
- Sometimes Verilog In creates schematics with objects that are off-grid.

  After you import the Verilog design, you might need to adjust the snap spacing to edit and manipulate these objects. For example, if symbols in a schematic have a different snap spacing from the schematic, the symbols might be off-grid. Refer to the following section for information about creating and editing symbols before you import the Verilog design.

Guidelines for Creating and Editing Symbols

Important

This section describes how to create or edit symbols using Virtuoso Schematic Editor before you import the Verilog design. These guidelines are for symbols that will be used in schematics created by Verilog In.

Important

If you do not follow these guidelines when creating or editing symbols, Verilog In might not create schematics, or might create schematics that are incorrect or off-grid. Also, nets might overlap symbols and symbol labels might overlap nets or net labels.

- To prevent extraction errors, make sure the snap spacing is 10 database units (dbu) or greater.

  When creating schematics, Verilog In takes the snap spacing for schematics and symbols from the property `xSnapSpacing` on the viewType schematic in the target library. Ideally, snap spacing should be an even number of dbu.

- To ensure that schematics are on-grid, check that the snap spacing of the target library and that of the symbols in the reference libraries are the same.

  If this is not possible, make sure the symbols in the reference libraries have a snap spacing that is a multiple of the snap spacing of the target library.

- To ensure that schematics are correct, check that the outer edges of all pin figures on each side of the symbol are on the same line (pin figures cannot be recessed from other pin figures) and are abutting the bounding box.

  The bounding box is the smallest box enclosing all pin figures and all shapes on the Device-Drawing Layer Purpose pair. Also, make sure that no symbol figures are beyond the pin figures on either side of the symbol shape. You should also specify a unique pin access direction on the pins of the symbol; otherwise, a net might be connected to a pin from a wrong direction.

- Snap spacing is set lower than 10 dbu in the `schematic` viewType in the target library.

- Snap spacing in the symbol is not a multiple of the snap spacing in the target.

Note: If the pins of a symbol are too close, the input/output pins of the cellview in which the symbol is placed may overlap (Figure 1-5). In such cases, the input/output pins are displaced and connected by name to the connecting net (Figure 1-6).

- To ensure that schematics are on-grid, make sure all marked lengths are a multiple of the snap space, including the pin-to-pin distances and the origin-to-pin distance.

  Verilog In places the origin of the Virtuoso Symbol on the snap grid. If all marked lengths are not multiples of the snap space, the schematic will be off-grid. Usually this is not a problem because the Symbol Editor automatically imposes this restriction. Problems might occur only if the snap space of the symbol has been changed during or after editing the symbol.

The following figures show sample symbols that were created for use with Verilog In. Although the first figure does not show pins on the top side of the symbol figure, you can place pins on any side of the symbol figure. The second figure shows some common mistakes made when creating symbols for use with Verilog In.

Figure 1-3 Sample Symbol Created for Verilog In Schematics

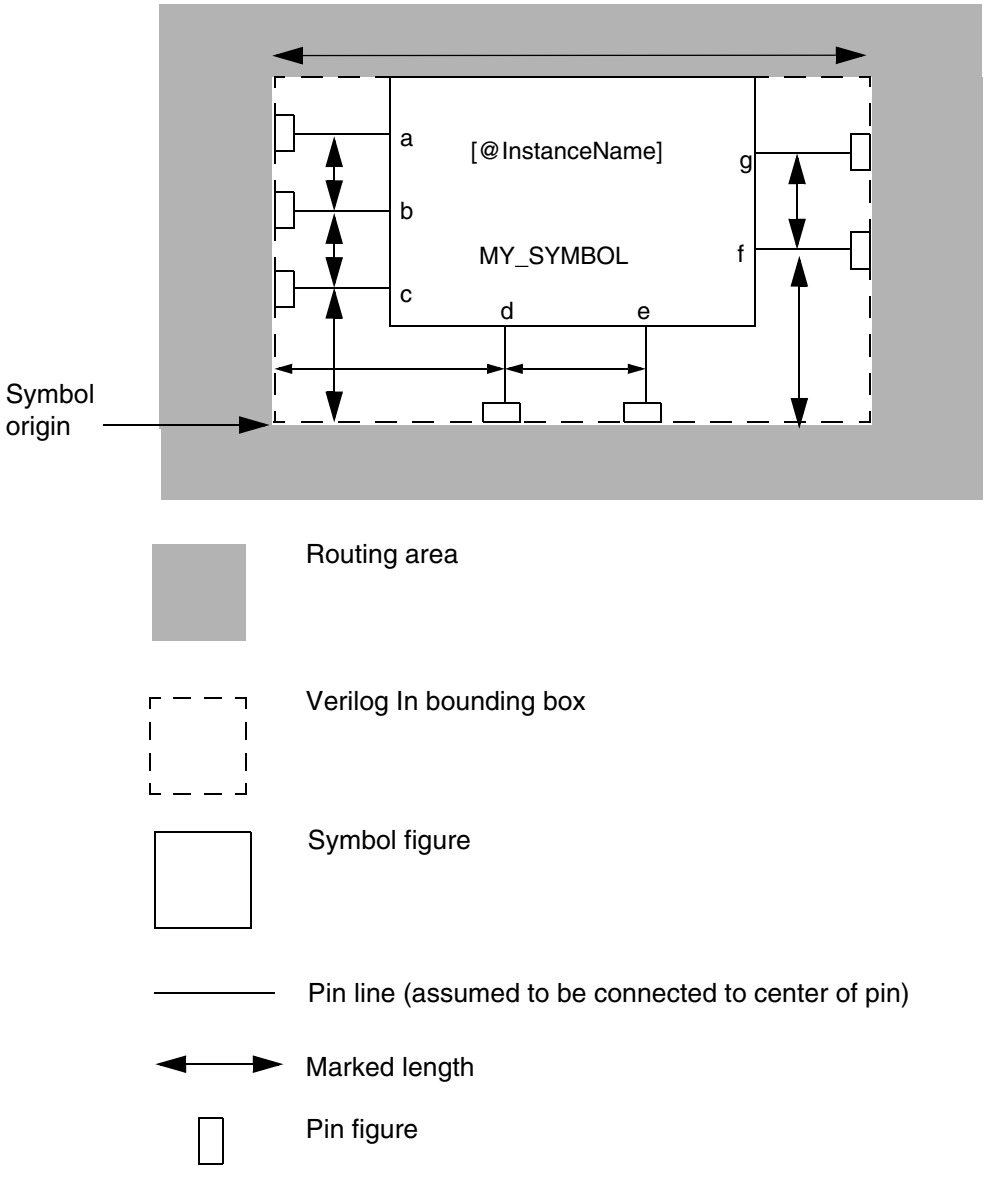
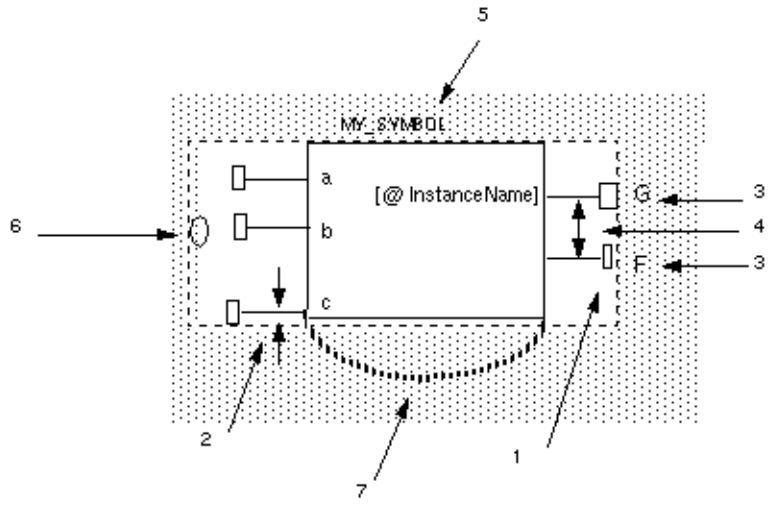


Figure 1-4 Common Mistakes Made When Creating Symbols



1. Pin figure is recessed. The outer edge of the symbol figure is not abutting the symbol bounding box.
2. Pin connection point is not a multiple of the snap space from the symbol origin.
3. Pin labels are outside the symbol's bounding box.
4. Distance between pin connection points is not a multiple of the snap space.
5. Symbol label is outside the symbol's bounding box.
6. Figure on a device drawing layer is outside the pin boundaries on that side.
7. Figure on a nondevice drawing layer is outside the bounding box.

Figure 1-5

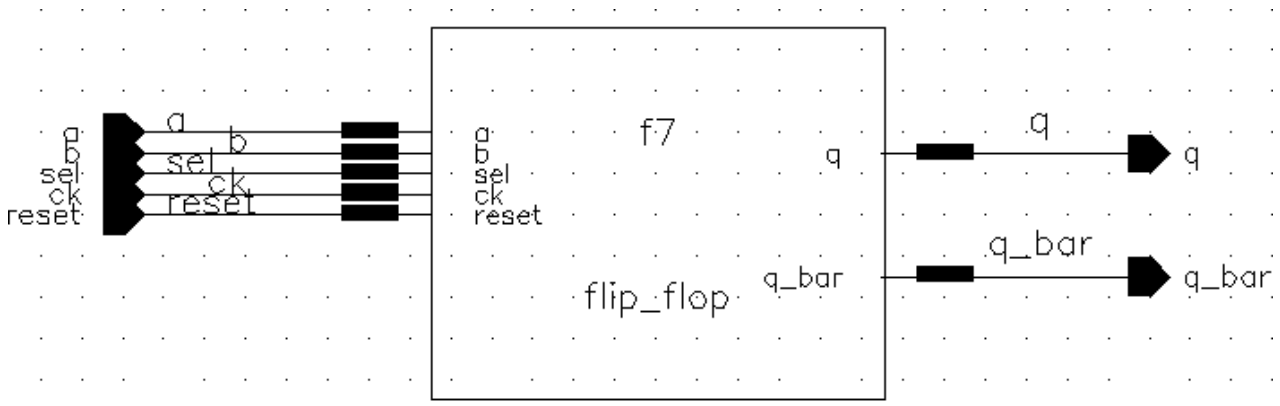
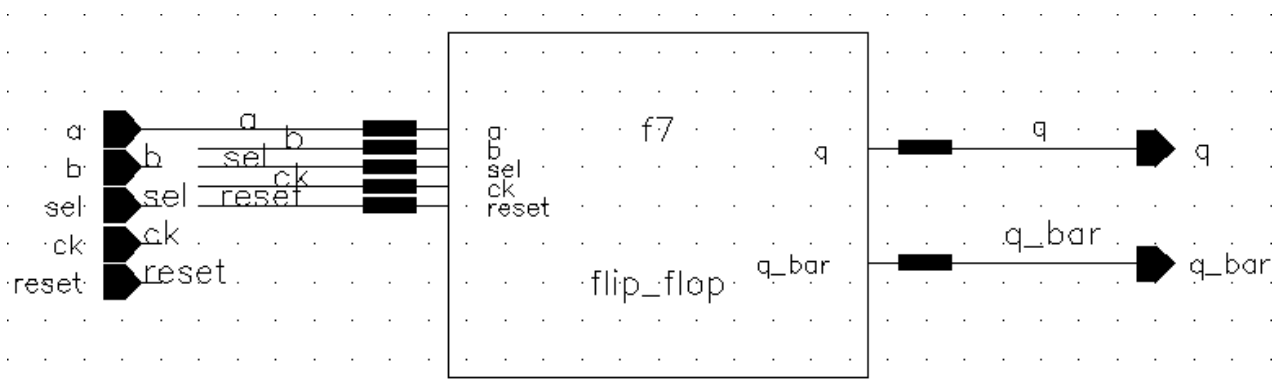


Figure 1-6



How Verilog In Imports Modules

This section describes how Verilog In imports modules.

- Verilog In considers a Verilog module and a Virtuoso Design Environment cell with the same name as the same entity.
- Verilog In creates a cell in the target library with the same name as the module and creates a symbol for the cell based on the constructs in the module.

  If the module is behavioral, Verilog In creates a functional cellview. If the module is structural, Verilog In creates a schematic cellview. Verilog In always creates a symbol, except when parameterized ports or duplicate ports are present in the module. Verilog In does not always create a functional or schematic cellview.
- If Verilog In identifies a module as structural and the module instantiates macromodules, Verilog In expands the macromodules in the schematic.

  If Verilog In identifies the module as behavioral, Verilog In does not expand the macromodules in the corresponding text view.
- Verilog In imports modules containing Verilog primitive strength specifications as textual cellviews, regardless of whether the strengths are Verilog keywords or integers.
- Verilog In does not import modules with multiple ports that have the same names.

Note: There is a limitation to the net name size of label which Verilog In can handle. If the number of bytes in the label for the net exceeds 16,384 bytes, then VerilogIn will exit with a message indicating the violating netname and its size in bytes. In this situation, the user is recommended to re-name or modify the net name accordingly such that size is within the maximum limit.

Using Reference Libraries to Import Incomplete Designs

Symbols from reference libraries are used to import incomplete designs.

Consider the following example:

If a module dff was imported earlier and a symbol exists for it in a reference library, you can import the module register (description given below) by giving the earlier imported library as a reference library without providing a definition for the module dff again.

```
module register(r, clk, data, ena, rst);
    output [7:0] r; input  [7:0] data; input clk, ena, rst;
    wire   [7:0] data, r;
    and a1(load, clk, ena);
    dff d0 (r[0], , load, data[0], rst),
    d1 (r[1], , load, data[1], rst), d2 (r[2], , load, data[2], rst),
    d3 (r[3], , load, data[3], rst), d4 (r[4], , load, data[4], rst),
    d5 (r[5], , load, data[5], rst), d6 (r[6], , load, data[6], rst),
    d7 (r[7], , load, data[7], rst);
endmodule
```

The search order for the symbol of an undefined module will be as follows:

First, the symbol view of the cell is searched for in the reference library.

Next, it is searched for in the destination library.

Once a symbol is found and it passes the selection criteria, it is used to generate structural views for instantiating module(s). A message is then generated saying that the module has not been defined and that the symbol is being used.

In case the symbol is not found or it does not pass the selection criteria, a functional view is created for the instantiating module(s).

Selection Criteria

The selection of a symbol is based on a consistency check with the first instance encountered.

1] A port number check is performed. For example, if the first instance encountered was:

```
Master1 A( w1, w2, w3 );
```

then the symbol should have at least 3 ports

2] A port name check is performed if the first instance encountered is connected by name. For example, if the instance encountered was:

```
Master2 A(.A(w1),.B(w2),.C(w3),.D(w4),.E(w5),.F(ww6));
```

then, the ports `A`, `B`, `C`, `D`, `E`, `F` should exist in the symbol.

The portOrder property on the symbol is used to resolve connectivity for implicitly connected instances. The portOrder property is added on all cell views created by Verilog In and is also used by the Virtuoso Design Environment Verilog netlister. In case a port order property is not found on the symbol, and an implicit instance is encountered Verilog In will terminate after displaying an error saying that it cannot resolve connectivity.

To handle bus ports, a port with the name A would match the symbol port with the name A <MSB:LSB>.

Escaped Name Mapping

This section describes how Verilog In handles escaped names. Escaped names are names preceded by a backslash and followed by a space. The advantage of using escaped names is that certain characters that are illegal in Verilog can be used in escaped names.

- The escaped names are mapped into legal names in the OpenAccess format and this information is stored in a file.
- Verilog In follows the Cadence standard name mapping scheme that is recognized by other Cadence tools.
- You can specify the name of the file in which you want the mapping information to be stored.
- The Verilog analyzer handles escaped names in certain cases. Information about these is not stored in the file.

Note: In 4.3.4, Verilog In used its own name mapping rules, but 4.4 onwards it uses the standard name mapping rules as described in *Chapter 7, Cadence Application Infrastructure User Guide*. The identifiers are mapped from the OA name space to the Verilog name space. You can use the Virtuoso's search and replace capabilities to customize the mapping on generated schematics.

How Verilog In handles parameters and defparams

Modules containing parameters and defparams will be treated as structural with the following exceptions.

Functional view will be created in the following cases.

- If the datatype of the value for parameter or defparam is other than decimal integer values, real values, and strings.
- If the parameter or defparam contains any arithmetic or logical expressions.
- If the parameter or defparam contains any identifier defined earlier by parameter statement.
- If parameters are used in the port declaration or net declaration.
- The inline parameter instantiation is used, but the source file containing the module definition is not given.

Note: The paramOrder property will not be added to the schematic or symbol imported and the user may add this manually. For more details about paramOrder, see "Adding Simulation Properties" in the "Netlisting" chapter of the *Virtuoso NC Verilog Environment User Guide.*

Data that is not Imported

This section describes data that is not imported by Verilog In.

- If Verilog In finds a cell in a reference library that has the same name as the module, Verilog In checks the names, the number of ports, and the port direction of both the module and the existing cell and does not import the module.

  If any of these do not match, Verilog In reports an error in the `verilogIn.log` file.

- If your design refers to library modules that have identical names, Verilog In imports only the first module found.

  Therefore, if you want to import both modules, you must change the name and the references of one module. You cannot use the Verilog-XL command line option `+liborder` as a work around because Verilog In does not support that option.

- If Verilog In finds a cell in the target library that has the same name and same view as the module, Verilog In does not import the module unless the *Overwrite Existing Views* option is `on`.
- If Verilog In finds the symbol view of a cell in the target library, it does not import the symbol view unless an appropriate value is set for the *Overwrite Symbol Views* option.
- If Verilog In identifies a module as structural, Verilog In does not import any comments inside or outside the module definition boundaries.

- If Verilog In identifies a module as functional, Verilog In does not import any text from within the module or text that follows the `endmodule` keyword, except for `'endcelldefine`.

  Compiler directives that precede the module definition are shown in the functional view. Except for `'endcelldefine`, compiler directives that follow the `endmodule` keyword are imported into different cells.

- Verilog In does not import modules that are split across multiple files.
- Verilog In imports modules with parameterized ports as functional views but does not create a symbol for them.

Problems that Might Occur

Before using Verilog In, be aware of the following problem that might result in loss of data.

If you try to backannotate a design after it has been imported, net names and module names might be different from the original Verilog design.

Verilog In Output Files

In addition to creating a Virtuoso schematic or a Virtuoso netlist, Verilog In creates a log file named `verilogIn.log`, which is described in the following section. Verilog In also creates the `verilogIn.map.table` file, which contains the original escaped names and the corresponding mapped names.

The verilogIn.log File

The `verilogIn.log` file is generated after the import process is complete. This file contains

- The module name
- Verification that the file was imported

  (If the module was not imported, the reason is given.)

- Any warnings related to the module

A `verilogIn.log` file contains entries, such as the following:

```
@(#)$CDS: ihdl version 6.1.6-64b 03/19/2014 21:16 (sjfnl116) $  Fri Mar 21
13:24:45 2014
Checked in functional view U_MUX_2_NONINV. UDP description found
Checked in functional view MUX21LB. Module is a cell
Checked in functional view U_FFD_P_RB_NOTI. UDP description found
Checked in functional view FD2. Module is a cell
Checked in functional view AN2. Module is a cell
Checked in symbol view_name
Checked in functional view counter_gnd. User specification
End of Logfile
```

The verilogIn.map.table File

The `verilogIn.map.table` file is created by Verilog In to store information about the original escaped names and the mapped names.

A `verilogIn.map.table` file contains entries such as the following:

```
VerilogIn Version 4.4.1 Wed Feb 12 17:18:30 1997
Original Name   Mapped Name
*****Map table for module x*****
\a?*b   a?*b
\A*B   A*B
```

2

Verilog In Standalone Mode

This chapter describes the following:

- Introduction on page 58
- Specifying Internal Options and Parameters on page 58

Note: For information on using SKILL functions of Verilog In, see *HDL Import and Netlist-to-Schematic Conversion SKILL Reference*.

Introduction

You can use the graphical user interface to specify the option and parameter settings for Verilog In. When working in standalone mode, however, you create files that specify the option and parameter settings and then you direct Verilog In to read those files. This section describes the internal options and parameter files and the command that directs Verilog In to read them.

Note: Cadence recommends that you use the graphical user interface to specify option and parameter settings.

You use the command `ihdl` to run Verilog In in standalone mode. To know how to use this command to import Verilog files into the Virtuoso design environment, run `ihdl` with the option `-help` or `-help3264`, as shown below. The option `-help3264` provides detailed assistance on using `ihdl`.

- `ihdl -help`
- `ihdl -help3264`

Specifying Internal Options and Parameters

This section describes how you specify and read in the internal option and parameters settings for Verilog In in standalone mode.

Starting in Standalone Mode

To start Verilog In in standalone mode, run the `ihdl` command with the relevant options and references,

The following is an example of how you run the `ihdl` command.

```
ihdl -f ihdl_files verilog_design_file
```

Where:

- `ihdl` is the executable name for Verilog In.
- `-f` directs the executable to read command arguments from an option file.
- `ihdl_files` is a file that contains the Verilog In options including the `-param` option. The `-param` option references the `ihdl_parameter` file, which contains the parameter settings.
- `verilog_design_file` is the Verilog file you want to import.

The ihdl_files File

As shown above, you specify the `ihdl_files` file in the startup command. The `ihdl_files` file contains option settings, such as those shown below.

```
-param ihdl_parameter
-v verilog_library_file.v
-y verilog_library_path
```

Verilog In Options

You can specify the following Verilog In options in `-f` files or as command line arguments.

Note: Although the examples are shown in uppercase letters, Verilog In options are not case sensitive; both `-y` and `-Y` are acceptable.

`-HELP`	Prints an online description about command line options.
`-IHDL_ALLOW_GLOBALS`	Allows global signals.
`-VERSION`	Prints the version number.
`-NOCOPYRIGHT`	Suppresses the printing of the copyright banner.
`+NO_PLACE`	Create a quick schematic without any placement and routing information.
`-NO_PORT_CHECK`	Does not check sizes.
`-OVER_DENSE`	Creates high-density schematics.
`-IGNOREEXTRAPINS`	Ignore extra pins in picking up reference symbols.
`-F <arg>`	Specifies a command line from a file.
`-V <arg>`	Specifies the name of the Verilog library file.
`-Y <arg>`	Specifies the name of the Verilog library directory.
`-PRECOMPILELIBRARY <arg>`	Specifies the pre-compiled library to be used for importing the design.
`-DESTIRLIB <arg>`	Specifies the name of the destination library where the pre compiled library will be created. If you specify more than one destination library, then only the first one will be used and the others will be ignored.
`-COMPILEONLY`	Specifies to prepare pre-compiled libraries only for libraries specified using -v and -y options and not import the entire design.

`-PARAM <arg>`	Specifies the name of a schematic parameter file.
`+DUMB_SCH`	Creates a quick place-only schematic with unrouted nets. The connectivity of these nets is indicated by name. To always create a place-only schematic, specify the `+DUMB_SCH` option or deselect the *Full Place and Route* check box in the *Schematic Generation Options* tabbed form. This is helpful when you use the *schematic* option for a design that contains a large number of gates or huge vector nets, and it takes several hours to complete the processing and requires a large amount of memory. In such a case, using `+DUMB_SCH` can significantly reduce the resource utilization.
`-NOSQUARE`	Does not square the schematic; that is, does not manipulate rows and columns of devices to convert a rectangular schematic into a square one.
`-MIN_CROSSOVERS`	Minimizes crossovers of nets.
`-FAST_LABELS`	Enables faster placement of labels. When this option is on, Verilog In labels segments at the midpoint and does not check for minimum overlap.
`+NOXTRSCH`	Does not extract the schematic; that is, does not find errors and warnings that have been written into the Cadence C-level database access format.
`-DEFINE <macro>`	Defines a macro from the command line.
`-cdslib <filename>`	Specifies the `cdslib` file.
`-hdlvar <filename>`	Name and location of the `hdl.var` file, which contains the variables and settings for the compiler, elaborator, and simulator.
`-VERBOSE`	Specifies whether to print detailed status messages while the schematic is being partitioned and routed.
`-NOEXTRANETS`	Specifies whether or not dummy nets will be present on unconnected pins of instances. This option only works for netlist view and not for schematic view.

The ihdl_parameter File

The `ihdl_files` file contains a call to the `ihdl_parameter` file. The `ihdl_parameter` file specifies the parameters for both the Verilog In and the Schematic Generation Options forms.

Example

```
-- Verilog In Form
dest_sch_lib := targetn
ref_lib_list := basic, sample, US_8ths
ignore_node_file := <ignore_node_file_name>
import_if_exists := 1
import_cells := 0
import_lib_cells := 0
structural_views := 5
schematic_view_name := schematic
functional_view_name := functional
netlist_view_name := netlist
symbol_view_name := symbol
overwrite_symbol := 1
log_file_name := ./verilogIn.log
map_file_name := ./verilogIn.map.table
work_area := <directory_name>
power_net := VDD!
ground_net := GND!
glob_sig_names := net1,net2
-- Schematic Generation Options Form
sheet_symbol := Asize
page_row_limit := 512
page_col_limit := 256
label_height := 12
line_line_spacing := 0.2
line_component_spacing := 0.5
density_level := 0
pin_placement := file, pin_placement_file
client := synthesis
alias_module := cds_alias
cont_assign_symbol := basic patch symbol
ref_sch_list := schematic, sch
pnr_max_inst := 20000
pnr_max_port := 5000
```

Important

The `ihdl_parameter` file must contain the following parameters: `dest_sch_lib`, `structural_views`, and `cont_assign_symbol`.

Parameters

The parameters that you specify in the `ihdl_parameter` file correspond to option fields on the Verilog In form or the Schematic Generation Options form as shown in the following table:

Tab	Field	Parameter
Import Options	*Target Library Name*	dest_sch_lib
	Reference Libraries	ref_lib_list
	HDL View Name	hdl_view_name
	Ignore Modules File	ignore_node_file
	Import Modules File	import_mod_file
	Import Structural Models As	structural_views
	Structural View Names	OA structural view names
	Log File	log_file_name
	Work Area	work_area
	Name Map Table	map_file_name
	Overwrite Existing Views	import_if_exists
	Overwrite Symbol	overwrite_symbol
	Verilog Cell Modules	import_cells
	Verilog Cell Modules	import_lib_cells
		enable_explicit_port_checking
		create_ifc_func_view
Global Net Options	*Power Net Name*	power_net
	Ground Net Name	ground_net
	Global Signals	glob_sig_names
Schematic Generation	*Sheet Symbol*	sheet_symbol
	Maximum Number Of Rows	page_row_limit
	Maximum Number Of Columns	page_col_limit
	Font Height	label_height

Tab	Field	Parameter
	Line To Line Spacing	line_line_spacing
	Line To Component Spacing	line_component_spacing
	Component Density	density_level
		alias_moldule
		clielnt
	Pin Placement	pin_placement
	Through CellView For Shorted Ports	cds_thru_symbol
	Create Snap Space Properties	create_snap_space_prop
	Reference Views for Inherited Connections	ref_sch_list **Note:** The corresponding variable of this parameter in `.cdsenv` is `refSchList`.
	Instances Less Than	pnr_max_inst
	Ports Less Than	pnr_max_port

In addition to these parameters, Verilog In provides some other parameters that you can use while importing Verilog designs. These parameters can be used to customize the default behavior of Verilog In using the corresponding environment variables. These variables are specified in the `.cdsenv` file, which can be accessed from the following path:

inst_dir`/tools/dfII/etc/tools/ihdl/.cdsenv`

The following table shows some of the entries in the `.cdsenv` file:

Tool Name	Environment Variable Name	Type	value	Range
ihdl	pin_master_basic_lib	boolean		nil
ihdl	pin_master_cells	string	ipin opin iopin	
ihdl	searchInReflib	boolean	t	nil
ihdl	maxNetNameLength	int	8000	nil

Tool Name	Environment Variable Name	Type	value	Range
ihdl	createFileLink	boolean	t	nil
ihdl	prefix_internal_net_name	string	" "	nil
ihdl	refSchList	string	" "	nil
ihdl	pnrMaxInst	int	20000	nil
ihdl	pnrMaxPort	int	5000	nil

The following table describes the parameters:

Parameter	Description
search_in_reflib	Determines whether to search the cells in the target and reference libraries before the cells are created. The search_in_reflib parameter can have either of the following values: ■ 1: Indicates that before creating the cells, these are searched in the target and reference libraries. ■ 0: Indicates that the cells are not searched in the target and reference libraries before the cells are created. The default value of the search_in_reflib parameter is 1. For more information, see the descriptions of the Target Library Name and Reference Libraries options. If both the search_in_reflib and import_if_exits parameters are set to 0, the cells are searched only in the target library and are created if these do not exist.

Parameter	Description
`max_net_name_length`	Specifies the maximum number of characters a net name can contain. The net name can contain the `8000` characters by default. A warning is generated if the number of characters in a net name exceeds `8000`. If the net name exceeds the number of characters you specify, the string is divided into multiple chunks. A shorter alias name is associated with each chunk. The maximum number of characters a chunk can contain is equal to the value of the *maxNetNameLength* variable. To obtain the complete net name, combine the different alias names.
`create_file_link`	If the value of this variable is nonzero, sets a soft link to the Verilog source file from which the Verilog design was imported into a functional view. The `create_file_link` parameter can have either of the following values: ■ `0`: Indicates that no pointer is set to the Verilog source file. ■ `1`: Indicates that a pointer is set to the Verilog source file. The default value of the `create_file_link` parameter is `0`.

Parameter	Description

`prefix_internal_net_name`	Specifies the prefix string for internal net names.
	
	In the schematic view, the default internal net name is `NeTt_<index>`and in the netlist view, the net names are `_LoNgNeTnAmE<index>` and `_NeTt_<index>`. However, when you generate a netlist from a design for which net names have leading underscores, the net names are escaped and require explicit name mapping. To avoid escaping net names and creating name maps, you can set the `prefix_internal_net_name` SKILL variable as shown below.
	
	In CIW:
	
	`envSetVal("ihdl" "prefix_internal_net_name" 'string "myprefix")`
	
	In the .cdsenv file:
	
	`"ihdl" "prefix_internal_net_name" 'string "myprefix" nil`
	
	After setting this SKILL variable, in the schematic view the net name will appear as:
	
	`myprefix_n_e_t_<index>`
	
	and in the netlist view as:
	
	`myprefix_l_o_n_g_n_e_t_n_a_m_e_<index>` and
	
	`myprefix_n_e_t_<index>`
	
	If you want to use the default net names, set the prefix argument (`myprefix`) in the `envSetVal` function to " ".

Net Expression Parameters

The mapping of GUI field with standalone paramfile parameter name is as follows:

Global Net Options Tab	
`netexpr_power_prop_name`	Net Expression Property Name for Power Net
`netexpr_ground_prop_name`	Net Expression Property Name for Ground Net

No flag required for this parameter	Create Net Expression **Note:** If some value is given to `netexpr_power_prop_name` and/or `netexpr_ground_prop_name`, the `Create Net Expression` is turned on internally in command line mode.
Schematic Generation Options Tab	
`conn_by_name_nets`	Connect By Name Nets

Verilog In Form Parameters

The following parameters correspond to option fields on the Verilog In form.

`dest_sch_lib`	Specifies the target library. Corresponding form field: Target Library Name `dest_sch_lib := <target library name>`
`ref_lib_list`	Specifies the reference libraries. Use a comma (,) to separate library names. Corresponding form field: Reference Libraries `ref_lib_list := basic, sample, US_8ths`
`hdl_view_name`	Specifies the view in the pre-compiled library which is used to find the IR for the cell while using pre-compiled libraries. Corresponding form field: HDL View Name `hdl_view_name := hdl`
`ignore_node_file`	Specifies the text file that contains the list of modules that you do not want to import. Corresponding form field: Ignore Modules File `ignore_node_file := <file path>` **Note:** This version of Verilog In does not support stop modules; therefore the `stop_node_file` parameter is no longer available.

`import_mod_file`	Specifies the text file, which lists the modules that you need to import. The default value of this parameter is `null`. Corresponding form field: Import Modules File `import_mod_file := <file path>`
`structural_views`	Specifies which views to create for structural modules. Corresponding form field: Structural Modules The valid values are 1 schematic only 2 netlist only 4 functional only 5 schematic and functional 6 netlist and functional `structural_views := 4`

`OA_structural_view_names`	Specifies which views to create in the target library.
	
	Corresponding form fields in: Import Modules As
	
	The following variables specify the view name to be used when creating the corresponding view, with their respective default view type:
	
	`functional_view_name := functional`
	`netlist_view_name := netlist`
	`schematic_view_name := schematic`
	`symbol_view_name := symbol`

`log_file_name`	Specifies a file that lists all error messages and log messages. Corresponding form field: Log File `log_file_name := <file path>`
`work_area`	Specifies a directory where Verilog In stores internal data. This parameter is useful when importing large designs and space in `/tmp` is limited. The default is `/tmp`. Corresponding form field: Work Area `work_area := <directory name>`

`map_file_name`	Specifies a file that lists original escaped names and corresponding mapped names. Corresponding form field: Name Map Table `map_file_name := <file path>`
`import_if_exists`	Specifies whether to import a module that has the same name as a module that already exists in the target library. When you set this value to 1, the module is imported. When you set this value to 0 (default), the module is not imported. Corresponding form field: Overwrite Existing Views `import_if_exists := 0` **Note:** `overwrite_option` works in the same way as `import_if_exists` option.
`overwrite_symbol`	Specifies whether to overwrite symbol of a module that already exists in the target library. By default, this parameter is set to 0 and symbols are not overwritten. Set this value to: ■ `1` to overwrite symbols created by Verilog In (symbols have the `createdBy` property set as `VerilogIn`). ■ `2` to overwrite symbols created by Text-to-Symbol generator or any other tool (symbols have the `createdBy` property set as any value other than `VerilogIn`). ■ `3` to overwrite all symbols. Corresponding form field: Overwrite Symbol Views `overwrite_symbol := 1`
`import_cells`	Specifies whether to import a Verilog HDL cell and the view type. When you set this value to 0 (default), the cell is imported as a symbol. When you set this value to 1, the cell is imported according to the value of the `import_lib_cells` option. Corresponding form field: Verilog Cell Modules `import_cells := 0`

`import_lib_cells`	When the `import_cells` option is set to 1, this option specifies a view type other than symbol for the `import_cells` option When this option is set to 0 (default), the cell is imported as a functional view. When set to 1, the cell is imported according to the value of the `structural_views` option. Corresponding form field: Verilog Cell Modules `import_lib_cells := 0`
`power_net`	Specifies the name of the global power signal in the Verilog design. You can specify only one power net name. All modules in the design recognize this name. Corresponding form field: Power Net Name `power_net := <user-defined power net name>` **Note:** If the "!" character occurs in the power net name, Verilog In does not add an extra "!" character.
`ground_net`	Specifies the name of the global ground signal in the Verilog design. You can specify only one ground net name. All modules in the design recognize this name. Corresponding form field: Ground Net Name `ground_net := <user-defined ground net name>` **Note:** If the ! character occurs in the ground net name, Verilog In does not add an extra "!" character.
`glob_sig_names`	Specifies that the nets in the Verilog design file to be treated as global. Corresponding form field: Global Signals `glob_sig_names := <user-defined global net name>`

`enable_explicit_port_checking`

If the module description contains explicitly defined empty ports, then:

- When the flag is set, a functional view is created.
- When the flag is not set, a symbol with a null port is created.

If the instance line contains undefined ports in case of explicit connections, then:

- When the flag is set, a functional view is created.
- When the flag is not set, a schematic is created.

```
enable_explicit_port_checking := 1
```

To enable this functionality when the import is done through GUI, you can set the `enableExplicitPortChecking` environment variable as shown below:

In CIW:

```
envSetVal("ihdl"
"enableExplicitPortChecking" 'boolean t)
```

In the .cdsenv file:

```
"ihdl" "enableExplicitPortChecking"
'boolean nil
```

`create_ifc_func_view`	Imports only the port information in any functional views that are created. All instance information is ignored.
	`create_ifc_func_view := 1`
	To enable this functionality when the import is done through CIW, you can set the `createIfcFunView` environment variable as shown below:
	In CIW:
	`envSetVal("ihdl" "createIfcFunView" 'boolean t)`
	In the .cdsenv file:
	`"ihdl" "createIfcFunView" 'boolean nil`

Schematic Generation Options Form Parameters

The following parameters correspond to option fields on the Schematic Generation Options form:

`sheet_symbol`	Specifies which sheet border size Verilog In applies when creating a multi-sheet schematic. The sheet borders reside in the `US_8ths` library. Also, ensure that the library where this symbol is present must be specified in the `ref_lib_list` parameter. When this value is `none` or if the symbol is not found in the reference library, Verilog In creates an infinite sheet schematic. Corresponding form field: Sheet Symbol `sheet_symbol := Asize`
`page_row_limit`	Specifies the maximum number of rows on each sheet. Usually, the maximum number of rows is expressed as a number between 1 and 1024. If missing, the limit is automatically set to `1024`. Corresponding form field: Maximum Number of Rows `page_row_limit := 512`

`page_col_limit`	Specifies the maximum number of columns on each sheet. Usually the maximum number of columns is expressed as a number between 1 and 1024. If missing, the limit is automatically set to `1024`. Corresponding form field: Maximum Number of Columns `page_col_limit := 1024`
`label_height`	Specifies the size of the font used for pin, wire, and instance labels. The value must be an integer greater than or equal to 1. Corresponding form field: Font Height `label_height := 10`
`line_line_spacing`	Specifies the spacing in inches between nets flowing in a channel. The value must be a real number in the range of 0.125 to 0.625. Corresponding form field: Line to Line Spacing `line_line_spacing := 0.2`
`line_component_ spacing`	Specifies the spacing in inches between a component and the nearest net flowing in a channel. The value must be a real number in the range of 0.125 to 0.625. Corresponding form field: Line to Component Spacing `line_component_spacing := 0.2`
`density_level`	Specifies the density of the schematic. The input must be an integer from 0 to 100, where 100 is the most dense and 0 is the least dense. Corresponding form field: Component Density `density_level := 0`
`cds_thru_symbol`	Specifies the library, cell, and view names for the cds_thru symbol. The default is `basic cds_thru symbol`. Corresponding form field: Through CellView For Shorted Ports `cds_thru_symbol := basic cds_thru symbol`

`alias_module`	While importing a Verilog module, if an instance is found with the same name as that supplied to the `alias_module` parameter, then Verilog In does not create a schematic for the module. Instead, it creates a functional view and a symbol for the module.
	`alias_module := cds_alias`
	The tool also reports the following message:
	`INFO (VERILOGIN-345): Checked in functional view top. Alias instance found.`

`client`	Distinguishes the text support functionality from the default Verilog In flow.
	`client := synthesis`
	`client := synth`
	`client := TextMode`

`cont_assign_Symbol`	Specifies the library, cell, and view names for the patch symbol. The default is `basic patch symbol`. Corresponding form field: Continuous assignment symbol `cont_assign_symbol := basic patch symbol`

`pin_placement`	Specifies the IO pin direction assignment for the schematic view.
	Corresponding form field: Pin Placement
	`pin_placement := all_sides`
	`pin_placement := left_and_right_sides`
	`pin_placement := file, <pin_placement_file_name>`

`create_snap_space_prop`	Specifies whether to disable the creation of snap space properties. Snap space can be defined as the minimum distance the cursor moves, in user units. The default distance (0.0625) is equal to half the default grid spacing distance. When this option is set to 0, then the x-y Snap Space properties are not created on the cellview. `create_snap_space_prop := 0`

`ref_sch_list`	Specifies comma-separated schematic or symbol views that the tool can look up for missing terminals on an instance line in the input Verilog file. If an instance line in the Verilog file has more number of terminals than the number of terminals on the identified instance symbol master, Verilog In uses the specified schematic/symbol views to look for the required additional terminals. The reference schematic/symbol views can contain terminals or nets with net expressions whose names match the names of the additional terminals on the instance line. The tool extracts the `netExpression` property name from those inherited terminals or nets. It then appropriately sets the `netSet` property on the instance for the missing terminals. The `netSet` property name is set using the `netExpression` property name obtained from the referenced schematic. The `netSet` property value is set from the net connected to the instance line in the Verilog file. Corresponding form field: Reference Schematic/Symbol Views For Inherited Connections `ref_sch_list := schematic, sch`
`pnr_max_inst`	Specifies the maximum number of instances that the design must have to trigger the generation of a place-only schematic. Corresponding form field: Instances Less Than `pnr_max_inst := 20000`
`pnr_max_port`	Specifies the maximum number of ports that the design must have to trigger the generation of a place-only schematic. Corresponding form field: Ports Less Than `pnr_max_port := 5000`

3

Verilog 2001 Support

This chapter describes the following:

- Use Model on page 78
- Verilog 2001 Constructs on page 78

Use Model

Verilog In has been enhanced to support Verilog IEEE 1364-2001 constructs. For other constructs, it does not create a schematic.

The `ncvlog` parser used by Verilog In supports the Verilog IEEE 1364-2001 constructs. The `ncvlog` parser version 08.20 is built into the DFII hierarchy. If you want to use a higher version of this parser from the IUS hierarchy, you can set the path as shown below:

```
set path =($path <IUS_BASE>/tools/bin )
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:'cds_root ncvlog'/tools/inca/lib
```

Verilog 2001 Constructs

Verilog In supports the following Verilog 2001 constructs:

- Signed Arithmetic
- Sized and Typed Parameters and Local Parameters
- Attributes
- Inherited Connections
- Named Parameter Assignment
- ANSI-C Style Port Declarations
- Combined Port and Type Declaration
- Indexed Part Selects
- Power Operator and Arithmetic Shift Operator

Signed Arithmetic

As per the Verilog IEEE 1364-1995 standard, the `integer` data type was signed while the `reg` and `net` data types were unsigned. The Verilog IEEE 1364-2001 standard has been enhanced to provide greater signed arithmetic capability, such as declaring signed reg and net data types.

The Verilog IEEE 1364-2001 standard uses the `signed` keyword to declare the `reg` and `net` data types, ports, and functions as signed data types as shown below:

```
wire signed [7:0] vector;
input signed [7:0] a;
```

To support signed net and port data types, a new boolean property, `verilogSignedDataType`, has been added by Verilog In. You can specify whether the net or port is signed or unsigned in the schematic or symbol view created by Verilog In. For example, if you have the following statement in the netlist:

```
wire signed [7:0] vector;
```

then Verilog In attaches the `verilogSignedDataType = t` property to the wire object.

Sized and Typed Parameters and Local Parameters

As per the Verilog IEEE 1364-2001 standard, you can explicitly declare the data type and size of parameters instead of determining these properties from the parameter value as shown below.

```
parameter verilog mux_selector = 0;
```

Traditionally, Verilog parameters are stored as simple name-value pairs under the hierarchical instance property, `verilog`. If the size and type are specified explicitly for a parameter then the name-value pair is modified to a name-list pair as follows:

`paramName1 = ("vin2001" "param" *signed type range value*)`

where,

`vin2001`	indicates that the property was imported by Verilog In under support of the Verilog 2001 constructs
`param`	indicates that this is a parameter
`signed`	value 1 for signed or value 0 for unsigned
`type`	any Verilog supported data type
`range`	in the format (lsb msb)
`value`	value of the parameter

The Verilog IEEE 1364-2001 standard also introduces a new type of module parameter called local parameter. A local parameter is similar to a parameter, except that it cannot be modified using a `defparam` statement or by ordered or named parameter value assignments in a module instance statement. Local parameters are declared using the `localparam` keyword as follows.

```
localparam signed [3:0] mux_selector = 0;
```

The format for specifying local parameters is similar to the parameter format described above except that the second value is `localparam` as shown below.

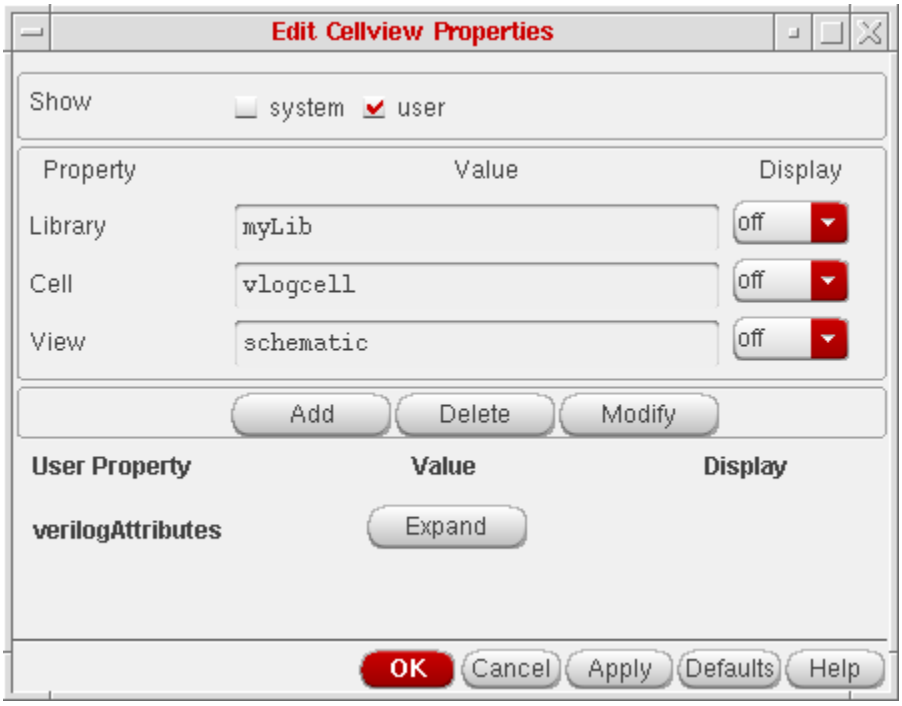
```
paramName1 = (”vin2001” ”localparam” signed type range value)
```

Attributes

The Verilog IEEE 1364-2001 standard allows you to specify properties for objects, statements, and groups of statements in the HDL source. These properties are used to control the operation or behavior of the tools. These properties are called attributes.

```
(* attributeName1=”attributeValue1”,
attributeName2=”attributeValue2”,attributeName3=”attributeValue3”) module
attribute_test(out,in,clk);
(* attrbuteName4="attributeValue4", attrbuteName5="attributeValue5" *) output
out; //module port  attributes
(* attrbuteName6="attributeValue6", attrbuteName7="attributeValue7" *) input in,
clk; //module port  attributes
(* attrbuteName8="attributeValue8", attrbuteName9="attributeValue9" *) wire m,n;
// net attributes
// instance attributes
(* attributeName12=” attributeValue12” *) and a2 ( m, n);
endmodule
```

For module, instance, port, and net, attributes are stored as hierarchical database properties on the cellview, instance, port, and net objects respectively and can be specified using the **verilogAttributes** option.



Inherited Connections

Verilog In is enhanced to support supply sensitivity specification, `netExpressions` including port `netExpression` and wire `netExpression` statements, and netSet properties. During import, Verilog In reads and creates `netExpression` and `netSet` properties on instances, ports and nets as applicable. The following topics explain the support for inherited connections in Verilog In.

Supply sensitivity specification

You can specify the supply port sensitivity information for power and ground supply as attributes of input, output, and inout ports. The attributes indicate the relationship between the port on which they are attached to the specified power port. Supply sensitivity information indicates the effective power and ground signals to use when an application needs to establish a logical connection to `1'b1` (power) or `1'b0` (ground), in a Verilog netlist.

The attributes `supplySensitivity` and `groundSensitivity` are used for specifying the sensitivity to the power supply port and ground supply port respectively.

```
module inv( a, y, powr, grnd) ;
    (* supplySensitivity = "powr", groundSensitivity = "grnd" *)
       input a ;
    (* supplySensitivity = "powr", groundSensitivity = "grnd" *)
       output y ;
```

As there is no direct way to verify that the specified port is in fact a supply port, therefore, a basic check is made to ensure that the port exists in the module port list.

The following two statements can be considered equivalent to the single statement as shown below.

```
(* powerSensitivity = "powr", groundSensitivity = "grnd" *)
    input a ;


(* powerSensitivity = "powr", groundSensitivity = "grnd" *)
    input b;
```

Equivalent single statement:

```
(* powerSensitivity = "powr", groundSensitivity = "grnd" *)
    input a, b;
```

Inherited connection specification (netExpression)

Net expression can be added either as attributes on a port declaration or to wires not associated with named ports. If the net expressions are added to wires, the connectivity is established through the hierarchy without explicit ports on the module declaration.

Port netExpression

A net expression added to a port indicates the name of the property for doing the connection look- up in the hierarchy, as well as a default value for the connection, if no explicit overwrite is found.

```
(* netExpr = "vdd(vdd_)" *) inout powr ;
```

The above declaration means that the input-output verilog port called `powr` has a net expression attached to it. The name of the property to search for in the hierarchy is “vdd_” and the default net name to connect to is the global net (in this case specified through an out of module reference) called `vdd(vdd_)`. In the case where the port is explicitly connected when the module is instantiated, the net expression is ignored, irrespective of the netSet properties defined at a higher level in the hierarchy.

Wire netExpression

A similar net expression declaration is also available for wires. Since there are no ports associated directly with a wire, the only way to overwrite the default value of an inherited connection, defined on a module wire, is by using a `netSet` property in a higher level of hierarchy.

```
(* netExpr = "clock(clck_)" *) wire CK ;
```

In the above example, to build the effective connectivity for the wire, `clck_`, the elaborator searches the hierarchy for a netSet named `clock`. In no netSet is found, then it defaults to the `clock(clck_)` net name.

Setting inherited connections (netSet)

NetSet statements are used to determine the value of the property that was specified in a net expression in the lower elements of a hierarchy to establish connectivity.

```
(* netSet = "vdd,vss" *)
(* vdd = "DVDD" , vss = "DVSS" *) moduleName
instanceName( portConnectivity));
```

In the above example, the values for the vdd and vss connections are assigned to the local net names DVDD and DVSS. This means that the net expressions located within the module

that is instantiated by this statement, or lower within the hierarchy of this module, are resolved to the DVDD and DVSS connections unless the expression is for a port that has an explicit connection.

For more information on inherited connections, see:

- Chapter 5, *Customizing the Hierarchical Netlister (HNL),* in the *Open Simulation System Reference* guide
- Chapter 9, *Netlisting,* in the *Virtuoso NC Verilog Environment User Guide*

Named Parameter Assignment

As per the Verilog IEEE 1364-1995 standard, you could modify the values of parameters declared within an instantiated module in only the following ways:

- Using the `defparam` statement, which allows assignment to parameters using their hierarchical names.
- Using module instance parameter value assignment, which allows values to be assigned inline during module instantiation. There are two forms of module instance parameter value assignment:
  - assignment by ordered list
  - assignment by name

The Verilog IEEE 1364-1995 standard supports only one form of module instance parameter value assignment - assignment by ordered list. The Verilog IEEE 1364-2001 standard includes module instance parameter value assignment by name. Parameter assignment by name consists of explicitly linking the parameter name with its new value. The parameter name is the name specified in the instantiated module. Verilog In has been enhanced to support named parameter assignment as shown below.

```
module ram (...);
    parameter WIDTH = 8;
    parameter SIZE = 256;
endmodule
module my_chip (...);
...
    ram #(8, 1023) ram1 (...);    // Parameter redefinition by position
    ram #(.SIZE(1023), .WIDTH(1000)) ram2 (...); // Parameter redefinition by name
    (named param assign)
...
 endmodule
```

ANSI-C Style Port Declarations

As per the Verilog IEEE 1364-1995 standard, the order of ports is defined within parentheses and the port declarations are listed after the parentheses. For tasks and functions, the parentheses list is omitted and the order in which the declarations have been specified is used to define the input/output order. The Verilog IEEE1364-2001 standard uses an updated syntax that is similar to the ANSI C language. When specifying the input and output declarations for modules, tasks, and functions, the declarations can be specified within parentheses to indicate the order of input and output.

For example, when declaring the input and output of a module, the following syntax is supported:

```
module mux8 (output wire [7:0] x,
    input wire [7:0] a,
    input wire [7:0] b,
    input wire enable);
    ...
    ...
endmodule
```

Verilog In parses both ANSI-C and non-ANSI-C style port declarations.

Combined Port and Type Declaration

For signals connected to the inputs or outputs of a module, you must declare the direction of the port and the type of the signal, such as net, reg, etc. As per the Verilog IEEE 1364-1995 standard you are required to specify these two declarations in two separate statements. The Verilog IEEE 1364-2001 standard provides a simpler syntax allowing you to combine the two declaration into one statement as shown below:

```
module mux8 (x, a, b, enable);
    output wire [7:0] x;
    input wire [7:0] a, b;
    input wire enable;
    ...
    ...
endmodule
```

Indexed Part Selects

In the Verilog IEEE1364-1995 standard, variable bit selects of a vector are permitted, but part-selects must be constant. You could not, for example, use a variable to select a specific byte out of a word.

The Verilog IEEE 1364-2001 standard adds a second type of part-select, called indexed part select. According to the standard, an indexed part select of a vector net, vector reg, integer variable, or time variable can be specified by providing a base expression, a width expression, and an offset direction, using the following syntax:

```
[base_expression +: width_expression]     //positive offset
[base_expression -: width_expression]     //negative offset
```

The base expression can be a constant or it can vary during simulation run time. The `width_expression` must be a constant. The offset direction indicates if the width expression is added or subtracted from the base expression.

```
module top;
    wire [5:0] in;
    wire [0:5] in2;
    wire [1:0] out2;
    BUF u2 (out2, in[4 -: 2]);
    BUF u1 (out2, in2[1 +: 2]);
endmodule
module BUF (out, in);
    input [1:0] in;
    output [1:0] out;
    buf (out[0], in[0]);
    buf (out[1], in[1]);
endmodule
```

Power Operator and Arithmetic Shift Operator

The Verilog IEEE 1364-2001 standard adds a power operator, represented by **. This operator performs exponential arithmetic.

```
module expr_delay (o, i);
    output o;
    input i;
      buf #(2 ** 3, 9 >>> 2) (p, i);
endmodule
```

Also, Verilog IEEE 1364-2001 supports two types of shift operators, the logical shift operators, << and >> and the arithmetic shift operators, <<< and >>>. Verilog IEEE 1376-1995 standard provides only the logical shift operator.

Constant expressions are allowed only in delays. Therefore, Verilog In supports these new operators for delays and creates functional view for all other cases.

A

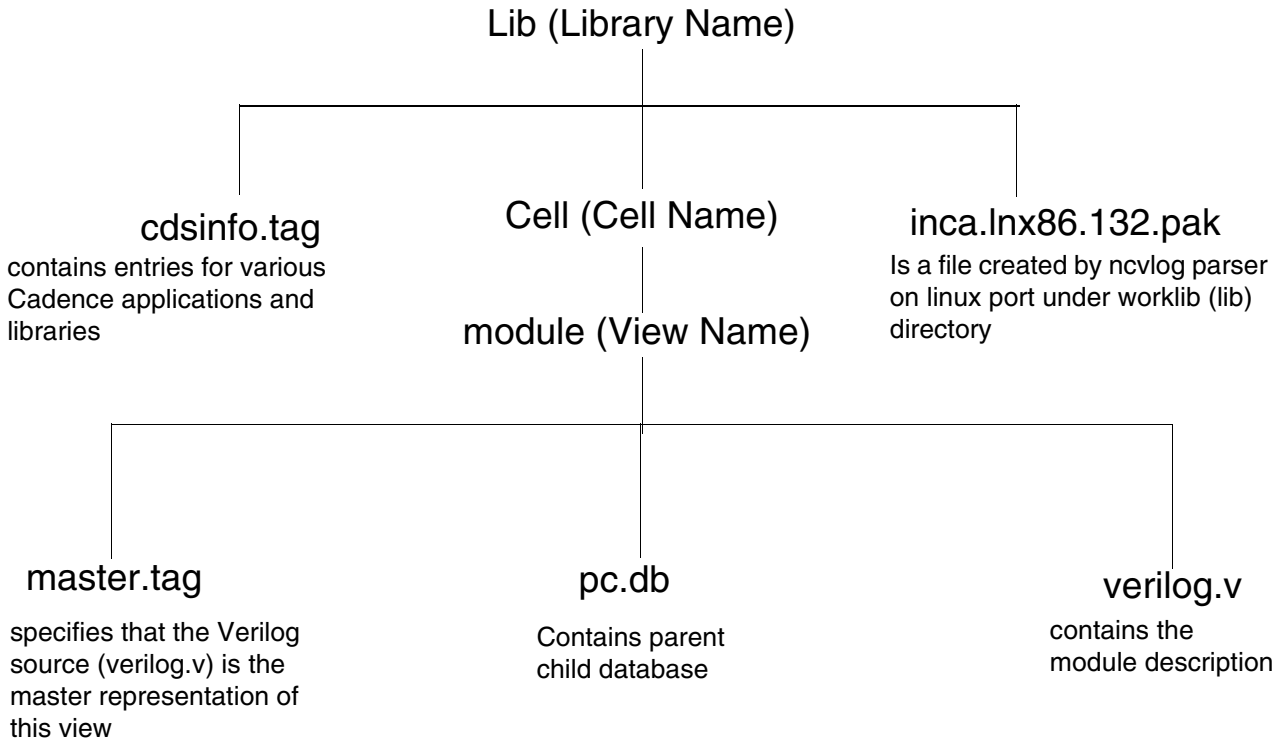
Pre-Compiled Libraries

This appendix describes the following:

- Introduction on page 88
- Creating Pre-Compiled Libraries on page 89
- Using Pre-Compiled Libraries on page 90
- Guidelines for Using Pre-Compiled Libraries on page 90
- Limitations on page 91
- Speeding up the Execution Process on page 91
- Speedup Results on page 92

Introduction

Verilog In uses the `ncvlog` parser to parse and translate the Verilog files into an Intermediate Representation (IR) that is stored in a `5.X` architecture library. During analysis, for each module in the Verilog file, `ncvlog` creates a cellview in the `5.X` library. The generated cellview has the following structure:



Verilog In compiles the source Verilog files in one go using ncvlog. A quick compilation without generation of any IR is performed on the library Verilog files to list all the modules in them. Whenever a library module is first referenced in the design, that module is compiled from the relevant Verilog library file and its IR is stored in the temporary work area.

If the same Verilog libraries are used over multiple runs of Verilog In it is much more advantageous to store the IRs of the Verilog files once for all in a pre-compiled library so that they can be used subsequently. In case of pre-compiled libraries, a Verilog library need not be compiled repeatedly to import designs.

Another instance where Pre-Compiled Libraries can be used effectively

When a destination IR library is specified, Verilog In compiles all libraries into the destination library in one go and then uses the pre-compiled library to import the design. Therefore in cases where the number of modules referenced from libraries is large, specifying a target IR library is advantageous even if the pre-compiled library is not required in future. In such cases, if the resources permit, you should try to make the destination IR library to point to the swap area of the machine for higher speed gain.

Creating Pre-Compiled Libraries

You specify the Pre-Compiled library name using `-destIRLib` option followed by the destination IR library name on the command line. If you specify more than one destination IR library names on the command line, then the first one is used and all the others are ignored.

You can also specify the destination library name using the Verilog In form in the *Target Compile Library Name* field. You can specify only one destination IR library name in this field.

Pre-compiled libraries can be created in two ways depending on your requirement:

1. Verilog In is used only to create the pre-compiled libraries and not to import the design. Later these pre-compiled IR libraries can be used while importing designs by specifying the `-compileOnly` option on the command-line or selecting the Compile Verilog Library only button in the Verilog In form.

   For example:

   ```
   ihdl -v abc.v -destIRLib Lib -compileOnly
   ```

   Verilog In translates all libraries specified by the `-v` and `-y` options to the target IR library specified. If `-v` or `-y` options have not been specified, Verilog In compiles and saves the IR for the design files specified on the command-line to the target IR library.

2. The other way is to import the design using `-v` and `-y` options along with a target IR library name. In this case, you must not select the *Compile Verilog Library only* check box.

   For example:

   ```
   ihdl -param param_file -v abc.v -destIRLib Lib design.v
   ```

   where *Lib* is the logical library name

   The design is imported and the IR for the libraries specified by the `-v` and `-y` options is saved permanently in the specified target IR library. The target IR library generated can

be used in future runs to import designs instead of the libraries specified by the `-v` and `-y` options in the first run.

Using Pre-Compiled Libraries

After creating the pre-compiled libraries, you can use them to import designs in place of original Verilog libraries. You specify the pre-compiled library name in the Verilog In GUI in the *Pre Compiled Verilog Library* field.

You can also specify the pre-compiled library name on the command-line using the `-preCompileLibrary` option followed by the logical name of the pre-compiled library. To specify multiple pre-compiled libraries on command-line, you need to repeat the `-preCompileLibrary` option.

For example,

```
ihdl -v xyz.v -preCompileLibrary Lib -preCompileLibrary Lib2 -y udp +libext+.v
BigDesign.v
```

Guidelines for Using Pre-Compiled Libraries

The guidelines for using pre-compiled libraries are:

- Any pre-compiled library to be used in importing must be defined in the `cds.lib` file.
- All IR cellviews have a link to the source file that points to the location from which it was compiled. The source file should not be moved from that location.
- Functional view are created from the source file while the symbol, the netlist, and the schematic views are created from the IR files. If the source file of a pre-compiled library is modified without creating the library again, then the functional view created matches the new description, but the symbol, netlist and the schematic views created match the older description. Therefore you must always create the pre-compiled library again each time the source file(s) from which it has been created is modified.

Limitations

The limitations in using pre-compiled libraries are:

- Pre-compiled library takes more space than a Verilog library.
- Although the physical location of a pre-compiled library can be changed, its logical name cannot be changed. If you want to change the logical name of pre-compiled library then you need to create it again.

Speeding up the Execution Process

- Always use Pre-Compiled libraries if they are available or if the same libraries are being used in multiple runs. You should create the pre-compiled libraries if they are not available.
- If you do not have any pre-compiled libraries and if, only a few modules are being referenced from the libraries, then you should use them as normal libraries. If the number of modules being referenced is large, you can specify a target IR library which should preferably be in the `/tmp` area or on a fast access disk area. This will reduce the execution time. The target IR library produced does not get deleted on completion of import and you need to remove it if it is not required.

Make sure you have enough swap or disk space for the target IR library.

Speedup in Case of Netlist View Creation

In this case, the flatter the design the better is the speedup.

Speedup in Case of Schematic View Creation

Use the fast label option if label positioning is not of much importance.

Speedup Results

Speed for design import for the netlist view has been increased. A system specification for which the speedup results were observed are:

```
Manufacturer    :Sun (Sun Microsystems)
System Model    :Ultra 5/10
Main Memory     :128 MB
Virtual Memory  :268MB
Number of CPUs  :1
CPU Type        :sparc
OS Name         :SunOS
OS Version      :5.5.1
Kernel Version  :SunOS Release 5.5.1 Version Generic_105428-01
                        [UNIX(R) System V Release 4.0]
```

The observed results are tabulated below:

Both Original and New Without Pre-Compiled Libraries for Netlist View

No. of modules in design	Size of Design	Previous time(sec)	New time(sec)	Speedup Factor
4	872K	3044	369	8.25
47	970K	2288	397	5.76
21	90K	260	237	1.09
371	3.32M	1924	1251	1.54
1073	3.04M	3294	2510	1.31
1	6.06M	25175	652	38.61

Another system specification for which the speedup results were observed are:

```
Manufacturer    :Sun (Sun Microsystems)
System Model    :Ultra 60
Main Memory     :1024 MB
Virtual Memory  :1.0 GB
Number of CPUs  :2
CPU Type        :sparc
OS Name         :SunOS
OS Version      :5.5.1
Kernel Version  :SunOS Release 5.5.1 Version Generic_103640-12
                        [UNIX(R) System V Release 4.0]
```

The observed results are tabulated below:

Both Original and New with Pre-Compiled Libraries for Netlist View

No. of modules in design	2391	13	9
No. of modules referenced from Verilog libraries	112/407	54/446	70/830
Size of Design	25M	4.69M	2.7M
Combined Size of Verilog libraries	1.4M	33K	1.4M
Previous time with -v option	1:55:02.5	1:01:29.0	1:10:49.4
New time with -v option	1:05:49.1	5:17.1	3:30.6
Speedup Factor	1.75	11.63	20.18
New time with pre-compiled library	54:21.7	5:07.8	1:17.1
Speedup Factor	2.12	11.99	55.12