cadence®

Verilog In User Guide

Product Version IC23.1
November 2023

Printed in the United States of America.

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

Contents

1

Introduction to Verilog In

Verilog In lets you import a Verilog Hardware Description Language (HDL) file into the Virtuoso Studio 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 topic is aimed at the designers of digital circuits and assumes that you are familiar with:

- The Virtuoso Studio 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 Studio Design Environment, notably Virtuoso Schematic Editor.

Prerequisites for Using Verilog In

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

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.
- Ensure that the design has been successfully compiled by the Verilog compiler.

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 Studio Design Environment, see Configuring the Virtuoso Studio Design Environment.

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 number 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

Related Topics

Classification of Modules in Verilog In

Verilog In Design Flow

When you use Verilog In with Virtuoso® Studio 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.

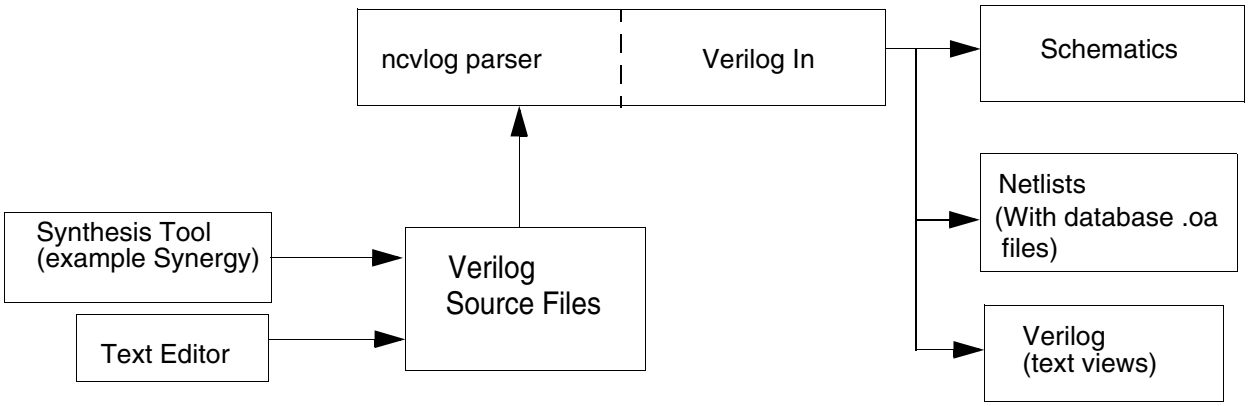
The Verilog In software is located in the Cadence software hierarchy at *install_dir*/tools/dfII/bin/ihdl, where *install_dir* is the directory where your Cadence software is installed.

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. In this case, Verilog In issues an error. Therefore, ensure that ncvlog is set in the path correctly. For details, see Verilog In with Verilog 2001 Support.

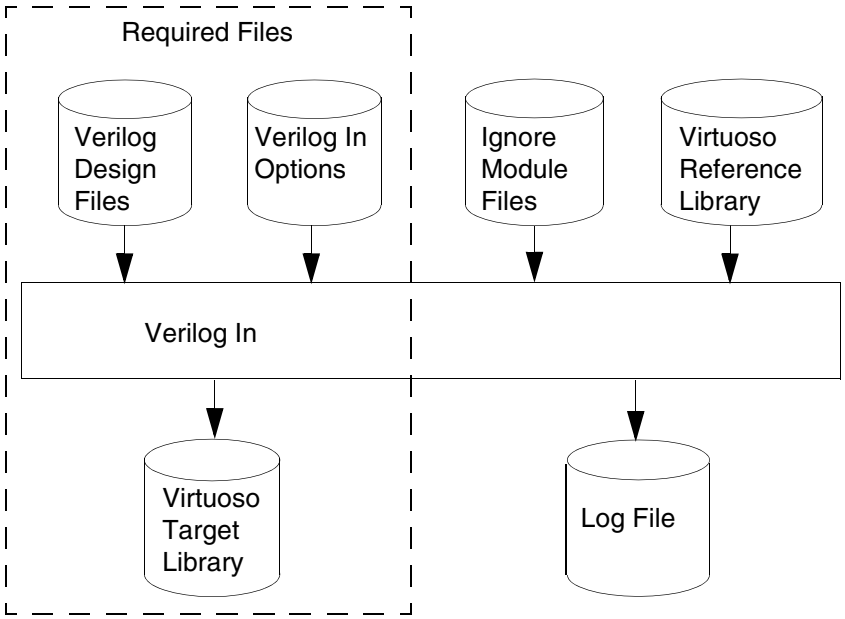
The following figure shows how to use Verilog In with other steps in the Verilog design process:



You can use Verilog In to do the following:

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

The following figure shows the files Verilog In uses and generates. The ‘Required Files’ are the files that you must specify in the Verilog In form.



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

Verilog In Command- Line Mode provides Information about running Verilog In in command-line mode. However, it is recommended that you use the Virtuoso Studio Design Environment user interface to run Verilog In.

Related Topics

Verilog In Command-Line Mode

Verilog In with Verilog 2001 Support

Starting Verilog In

To start Verilog In:

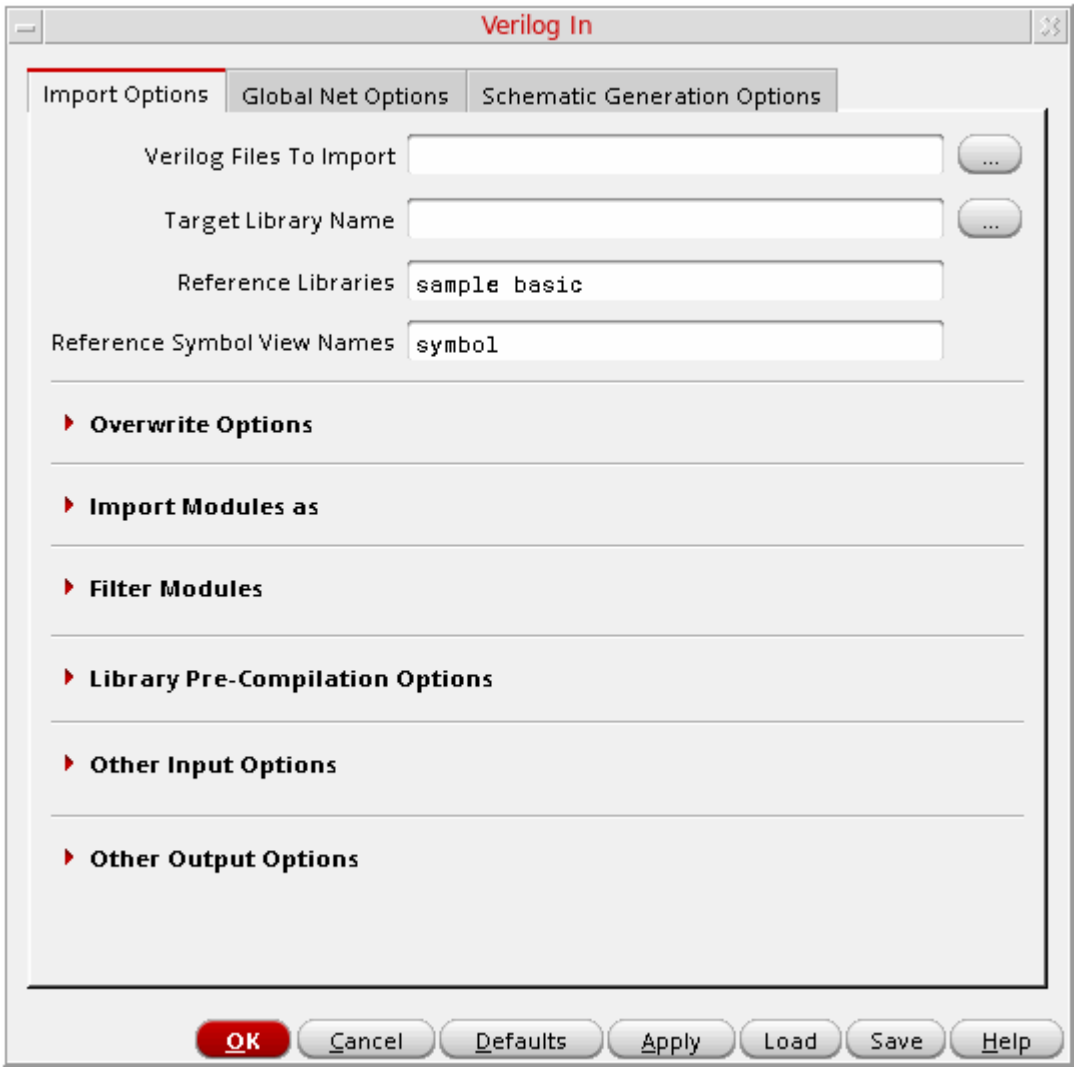
1. In an xterm window, type the following command:

   ```
   virtuoso &
   ```

   The Command Interpreter Window (CIW) appears.

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. Open the Verilog In form.

   The *Import Options* tab appears.

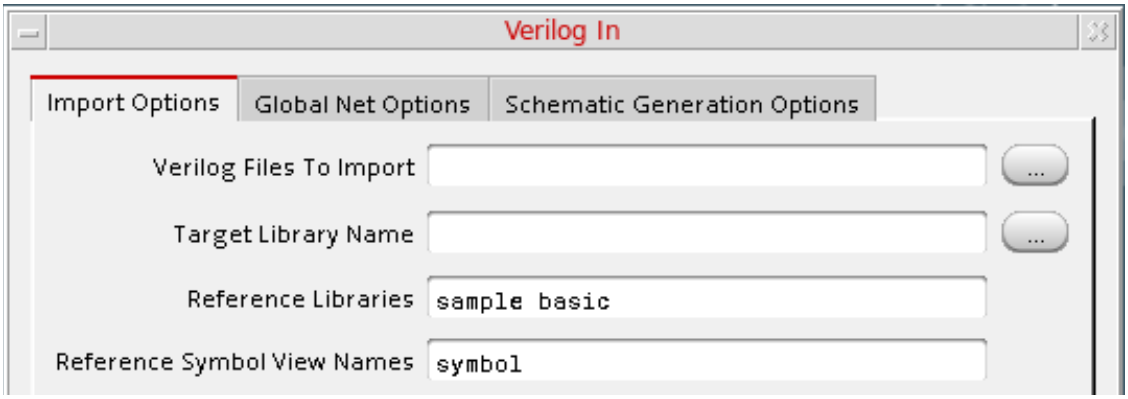
2. Specify the files you want to import in the Verilog Files to Import field.

   You can specify multiple files with paths.

3. Specify the library where the file must be imported in the Target Library Name field.

4. Specify 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` is imported to the library `myLib` and uses the reference libraries `sample` and `basic`.



5. Specify any other options you need by expanding the relevant sections.

   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.

6. If required, set the global net options as follows:

   a. Click the Global Net Options.

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

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

7. If required, set the schematic generation options as follows:.
   a. Click the Schematic Generation Options.
   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.
8. Click *OK* or *Apply*.

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

For details on the fields in all tabs, see Verilog In Form.

Saving Import Options in Verilog In

To save import options in Verilog In:

1. Specify all the required options in the Verilog In form.
2. Click *Save*.

   The Save Verilog In Option form appears.
3. Specify a valid filename.

   You can select an existing file or specify a new filename.
4. Click *Save*.

   The Save Verilog In Options form closes.
5. In the Verilog In form, click *OK* or *Apply*.

Related Topics

Verilog Files to Import

Target Library Name

Reference Libraries

Verilog In Form

Loading Import Options in Verilog In

To load import options in Verilog In:

1. In the Verilog In form, click *Load*.

   The Load Verilog In Options File form appears.
2. Specify a valid filename that contains previously saved import options.
3. Click *Open*.
4. In the Verilog In form, click *OK* or *Apply*.

The Load Verilog In Options File form closes and all import options are loaded.

Classification of Modules in Verilog In

This topic describes the following details:

- Classification of modules
- Guidelines for modifying your design
- Guidelines for creating schematic symbols
- Imports of modules in Verilog In
- Escaped name mapping
- Parameters and defparams handling by Verilog In
- Reasons behind failure of importing data
- Miscellaneous problems

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 Studio Design Environment library. Verilog In checks each module and decides if it is one of the following:

- 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. See Guidelines for Design Modification in Verilog In.

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

Related Topics

Guidelines for Design Modification in Verilog In.

Guidelines for Design Modification in Verilog In

The following topic describes how Verilog In imports data and how to modify the design before and after importing 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 Studio 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 Studio 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, such as 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. See Guidelines for Creating and Editing Symbols for information about creating and editing symbols before you import the Verilog design.

Related Topics

Guidelines for Creating and Editing Symbols

Guidelines for Creating and Editing Symbols

This topic describes how to create or edit symbols using Virtuoso Schematic Editor before you import the Verilog design. These guidelines are for symbols that are 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 `schematic` viewType in the target library. Ideally, snap spacing must 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 meaning 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 must 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.

  If the pins of a symbol are too close, the input/output pins of the cellview in which the symbol is placed may overlap. In such cases, the input/output pins are displaced and connected by name to the connecting net.

- 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 is 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.

Significance of Reference Libraries for Importing 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 is as follows:

1. First, the symbol view of the cell is searched for in the reference library.
2. 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 modules. A message is then generated specifying 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 modules.

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

- A port number check is performed.

  For example, in case of the following instance that is detected by the tool, the symbol must have at least three ports.

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

- A port name check is performed if the first instance encountered is connected by name. For example, in case of the following instance that is detected by the tool, the ports `A, B, C, D, E,` and `F` must exist in the symbol.

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

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

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

Escaped Name Mapping in Verilog In

This topic 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 Cadence Application Infrastructure. The identifiers are mapped from the OA name space to the Verilog name space. You can use the search and replace capabilities of Virtuoso to customize the mapping on generated schematics.

Parameters and Defparams in Verilog In

Modules containing parameters and defparams are treated as structural. However, functional views are created for the following exceptional 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 is 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.

Exceptions in Data Import by Verilog In

This topic 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.

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.

Output Files Created by Verilog In

In addition to creating a Virtuoso schematic or a Virtuoso netlist, Verilog In creates a log file named `verilogIn.log`. 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. The contents of this file include:

- The module name
- Verification that the file was imported

  (If the module was not imported, the log describes the reason.)
- Any warnings related to the module

A sample `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 sample `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 Command-Line Mode

You can use the graphical user interface to specify the option and parameter settings for Verilog In. When working in command-line 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 command-line mode. To know how to use this command to import Verilog files into the Virtuoso Studio Design Environment, run `ihdl` with the option `-help` or `-help3264`. The option `-help3264` provides detailed help on using `ihdl`. For example:

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

Starting Verilog In in Command-Line Mode

To start Verilog In in command-line mode:

➡ Run the `ihdl` command in terminal window with the relevant options and references.

The following example shows how to run the `ihdl` command:

```
ihdl -f ihdl_files verilog_design_file
```

Here:

- `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.

Components of the ihdl Command

The ihdl command includes three main components, the `ihdl_files` file, `-f` options and the `ihdl_parameter` File.

The ihdl_files File

You specify the `ihdl_files` file in the startup command. The `ihdl_files` file contains option settings, as follows:

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

Options Specified with -f in the ihdl Command

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

Verilog In options are not case sensitive. For example, `-help` and `-HELP` are considered the same.

`-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`	Creates a schematic without any placement and routing information.
`-NO_PORT_CHECK`	Does not check sizes.
`-OVER_DENSE`	Creates high-density schematics.
`-IGNOREEXTRAPINS`	Ignores 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 gets created. If you specify more than one destination library, then only the first one is used and the others are ignored.
`-COMPILEONLY`	Specifies that pre-compiled libraries must be compiled 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 schematic that does not indicate nets or connectivity by name.
`-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>`	Specifies the 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 are 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 `dest_sch_lib`, `structural_views`, and `cont_assign_symbol` parameters.

Parameters Specified in the ihdl_parameter File

The parameters that you specify in the `ihdl_parameter` file correspond to option fields in the Verilog In 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

Tab	Field	Parameter
	Font Height	label_height
	Line To Line Spacing	line_line_spacing
	Line To Component Spacing	line_component_spacing
	Component Density	density_level
	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 Environment Variable: `refSchList`
	Instances Less Than	pnr_max_inst
	Ports Less Than	pnr_max_port

Customization of Verilog In Defaults Using the .cdsenv File

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

Tool Name	Environment Variable Name	Type	value	Range
ihdl	maxNetNameLength	int	8000	nil
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 that correspond to the environment variables available in the.cdsenv file:

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 Target Library Name and Reference Libraries. 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` variable as follows:
	
	In CIW:
	
	`envSetVal("ihdl" "prefix_internal_net_name" 'string "myprefix")`
	
	In the `.cdsenv` file:
	
	`"ihdl" "prefix_internal_net_name" 'string "myprefix" nil`
	
	Setting this environment variable in the schematic view displays the net name as follows:
	
	`myprefix_n_e_t_<index>`
	
	Setting this environment variable in the netlist view displays the net name as follows:
	
	`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 " ".

Related Topics

Target Library Name

Verilog In Options and Parameters Specified in the ihdl_parameter File

The following table describes the parameters defined in `ihdl_parameter` file and their corresponding form fields.:

Parameters	Description
Import Options	These parameters correspond to option fields on the *Import Options* tab.
`dest_sch_lib`	Specifies the target library. Form Field: Target Library Name `dest_sch_lib := <target library name>`
`ref_lib_list`	Specifies thForm fielde reference libraries. Use a comma (,) to separate library names. 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. 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. 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`. Form Field: Import Modules File `import_mod_file := <file path>`

Parameters	Description
`structural_views`	Specifies which views to create for structural modules. Form Field: Structural Modules Valid values are: ■ schematic only ■ netlist only ■ functional only ■ schematic and functional ■ netlist and functional `structural_views:= 4`

`OA_structural_view_names`	Specifies the views to be created in the target library. The default values are `functional`, `netlist`, `schematic`, and `symbol` respectively.
	Form Field: Import Modules As
	`functional_view_name := functional`
	`netlist_view_name := netlist`
	`schematic_view_name := schematic`
	`symbol_view_name := symbol`

`log_file_name`	Specifies the file that lists all error messages and log messages. Form Field: Log File `log_file_name := <file path>`
`work_area`	Specifies the 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`. Form Field:Work Area `work_area := <directory name>`
`map_file_name`	Specifies the file that lists original escaped names and corresponding mapped names. Form Field: Name Map Table `map_file_name := <file path>`

Parameters	Description
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 its default 0, the module is not imported. Form Field: Overwrite Existing Views `import_if_exists := 0` **Note:** The overwrite_option works in the same way as the 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 the following values: ■ 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. 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 its default 0, 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. Form Field: Verilog Cell Modules `import_cells := 0`

Parameters	Description
`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 its default `0`, 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. Form Field: Verilog Cell Modules `import_lib_cells := 0`
Global Net Options	These parameters correspond to option fields on the *Global Net Options* tab.
`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. Form Field: Power Net Name `power_net := <user-defined power net name>` If the "!" character exists 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. Form Field: Ground Net Name `ground_net := <user-defined ground net name>` If the "!" character exists 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 must be treated as global. Form Field: Global Signals `glob_sig_names := <user-defined global net name>`

Parameters	Description

enable_explicit_port_checking	If the module description contains explicitly defined empty ports:
	■ 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:
	■ 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 follows:
	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 follows:
	In CIW:
	`envSetVal("ihdl" "createIfcFunView" 'boolean t)`
	In the `.cdsenv` file:
	`"ihdl" "createIfcFunView" 'boolean nil`

Parameters	Description
Schematic Generation Options	These parameters correspond to option fields on the *Schematic Generation Options* tab.
`sheet_symbol`	Specifies which sheet border size Verilog In applies when creating a multisheet schematic. The sheet borders reside in the `US_8ths` library. When this value is `none`, Verilog In creates an infinite sheet schematic. 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`. 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`. 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`. 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`. Form Field: Line to Line Spacing `line_line_spacing := 0.2`

Parameters	Description
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. 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. 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. Form Field: Through CellView For Shorted Ports cds_thru_symbol := basic cds_thru symbol
cont_assign_Symbol	Specifies the library, cell, and view names for the patch symbol. The default is basic patch symbol. Form Field: Continuous assignment symbol cont_assign_symbol := basic patch symbol
pin_placement	Specifies the IO pin direction assignment for the schematic view. 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 the creation of snap space properties must be enabled or disabled. 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

Parameters	Description
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 or symbol views to look for the required additional terminals. The reference schematic or 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. 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. 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. Form Field: Ports Less Than `pnr_max_port := 5000`

Net Expression Parameters

The following table shows the mapping of GUI field with `paramfile` command-line parameter name:

Parameter	Form Field
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

3

Verilog In with Verilog 2001 Support

This topic explains the Verilog 2001 constructs for which Verilog In can create schematics. Verilog In supports 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 follows:

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

Verilog In supports the following Verilog 2001 constructs:

- Signed Arithmetic
- Sized and Typed Parameters and Local Parameters
- Attributes in HDL Source
- 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 follows:

```
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, consider that you have the following statement in the netlist:

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

In such a case, 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*)`

The following list describes various components of a name-list pair:

- `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` specifies the value `1` for signed or `0` for unsigned
- `type` specifies any Verilog supported data type
- `range` indicates the range in the lsb or msb format
- `value` specifies the 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 except that the second value is `localparam` as follows:

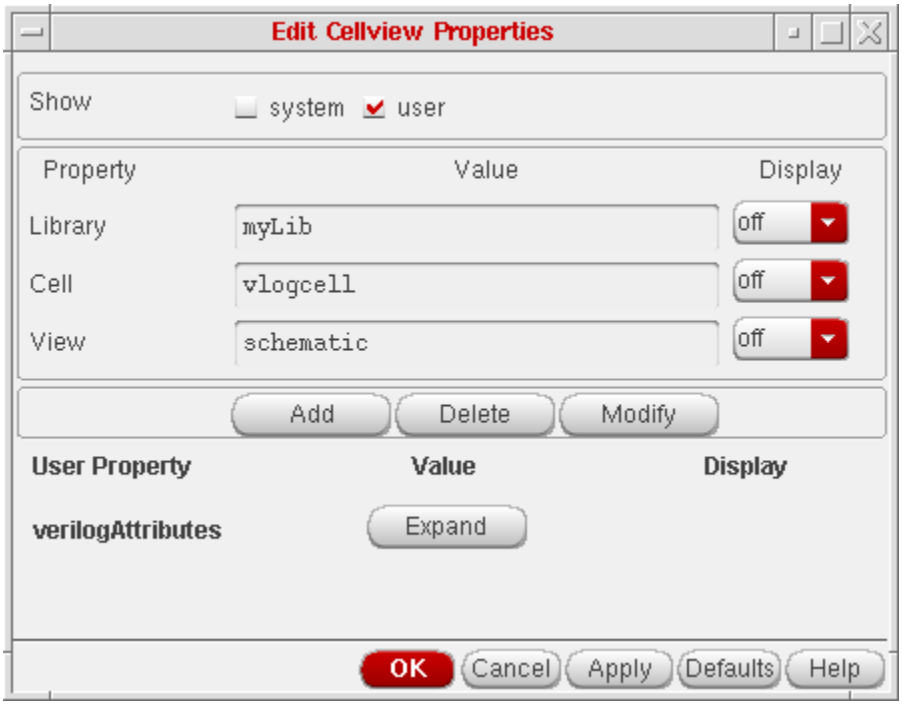
```
paramName1 = (”vin2001” ”localparam” signed type range value)
```

Attributes in HDL Source

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 in the Edit Cellview Properties form.



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.

Inherited Connections Setup(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 Customizing the Hierarchical Netlister (HNL) ,and Netlisting.

Named Parameter Assignment

As per the Verilog IEEE 1364-1995 standard, you can 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 follows.

```
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

Verilog In Form

The Verilog In form is organized in three tabs for import options: *Import Options*, *Global Net Options,* and *Schematic Generation Options.*

Tab	Description
Import Options	Lets you specify the options available in Verilog In to import the schematic.
Global Net Options	Lets you specify the global nets and net expressions of the schematic imported by Verilog In.
Schematic Generation Options	Lets you specify the format of the schematic imported by Verilog In.

Import Options

The following table describes the fields available on the *Import Options* tab of the Verilog In form.

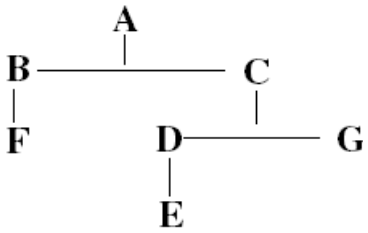
Field	Description
Verilog Files to Import	Lets you select the names of the Verilog files that contain the complete descriptions to import. You can type or select multiple files separated by a space. To select the files, click the corresponding *Browse* button. In the Select File form, browse and select the files you want to import. To select multiple files, hold down the `Ctrl` key and select the files.

Field	Description
Target Library Name	Specifies the Virtuoso Studio 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, else, Verilog In creates a new target library and places the imported cells in it. *Browse* opens the Library Browser form. Clicking a library name in Library Browser puts that library name in the *Target Library Name* field.
Reference Libraries	Lets you reference the libraries that contain Virtuoso Studio 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 modules. For more details on symbol selection, see Using Reference Libraries to Import Incomplete Designs.
Reference Symbol View Names	Lets you reference multiple symbol view names and find the instance master in the reference library.

Field	Description
Overwrite Options	This section specifies overwrite options for the cellviews to be generated by the import process.
Overwrite Existing Views	Overwrites existing views with a new version if the module to be imported already exists as a cell in the target library. 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	Overwrites the specified symbol view with a new symbol if the symbol of the module to be imported already exists in the target library. 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. To view the `createdBy` property of any symbol, open it in *Virtuoso Symbol Editor L* and 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. ■ *All*: Overwrite all the symbols existing in the target library and replace with those imported by Verilog In.

Field	Description
Import Modules as	This section provides the options to import the modules.
Structural Modules	Specifies the views that must be created for structural modules by Verilog In. Possibe values: ■ `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* tab 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.

Field	Description
Verilog Cell Modules	Specifies 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	Specifies the name for the schematic view to be created in the target library. The default view is `schematic`.
Netlist View Name	Specifies the name for the netlist view to be created in the target library. The default view is `netlist`.
Functional View Name	Specifies the name for the functional view to be created in the target library. The default view is `functional`.
Symbol View Name	Specifies the name for the symbol view to be created in the target library. The default view is `symbol`.

Field	Description
Filter Modules	This section specifies options to filter modules in the schematic.
Ignore Modules File	Specifies the text file that lists modules which you do not want to import. Modules underneath these modules in the design hierarchy are imported, unless they too are listed in this file. 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	Specifies 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. 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	This section provides the options to specify pre-compiled libraries.
Pre Compiled Verilog Library	Specifies 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.

Field	Description
HDL View Name	Specifies 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 is saved. The default view name is `hdl`.
Target Compile Library Name	Specifies the `5.X` library where the Intermediate Representation (IR) produced for the Verilog libraries specified using the `-v` and `-y` options that are located. You can also click the *Browse* button to open the library browser and select a target library.
Compile Verilog Library Only	Compiles pre-compiled libraries without importing the design.
Other Input Options	This section lets you specify additional input options.
-f Options	Specifies 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	Specifies the names of the Verilog files needed to compile the design. The reference design files can be given as `-v` options.
-y Options	Specifies the name of the Verilog library file.
Library Extension	Specifies 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	This section lets you specify additional output options.
Log File	Specifies the file that logs the import status of each module being imported. For more information about the log file, see Output Files Created by Verilog In. The default log file is `./verilogIn.log`.
Work Area	Specifies 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.

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

Process Technology File	Specifies 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 in one of the following ways:
	
	■ In the CIW, type
	
	`envSetVal("ihdl" "processTechFile" 'boolean t).`
	
	■ In the `.cdsenv` file, specify
	
	`ihdl processTechFile boolean t/nil.`

Global Net Options

The following table describes the fields available on the *Global Net Options* tab of the Verilog In form. Use these options to specify global nets and net expressions of the schematic imported by Verilog In.

Field	Description
Global Nets	This section provides options to specify names for a global power net, a global ground net, and other global signals in the schematic.
Power Net Name	Specifies 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.
Connect Power Net by Name	Connects the power net by the specified power net name. Use this option to import a logical netlist with the power net connected by name, instead of a physical wire. 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.
Ground Net Name	Specifies 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.
Connect Ground Net by Name	Connects the ground net by the specified ground net name. Use this option to import a logical netlist with the ground net connected by name, instead of a physical wire. 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.
Global Signals	Specifies 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. All pins and nets, except power and ground pins and nets, have the default signal name set to `signal`. For power pins and nets, the default signal name is `supply`, and for ground pins and nets, the default signal name is `ground`.

Field	Description
Connect Global Signals by Name	Connects global signals by the specified global signals name. Use this option to import a logical netlist with the global signals connected by name, instead of a physical wire. 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	This section lets you specify options to create net expressions. If you select this option, you must specify valid values for *Property Name for Power Net* and *Property Name for Ground Net*.
Create Net Expression	Lets you translate tie highs and tie lows so that net expression can be 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	Specifies a different net expression property name for power nets. The default is `vdd`.
Property Name for Ground Net	Specifies a different net expression property name for ground nets. The default is `gnd`.

Schematic Generation Options

The following table describes the fields available on the *Schematic Generation Options* tab of the Verilog In form. Use these options to specify the format of the schematic imported by Verilog In.

Field	Description
Full Place and Route	Specifies placement and routing options for the schematic. Lets you 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. ■ 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.

Field	Description
Instances Less Than	Specifies 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. The default value is `20000`. Environment variable: `pnrMaxInst`. For information on the corresponding parameters for the `ihdl_parameter` file, see Verilog In Command-Line Mode.
Ports Less Than	Specifies 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. These fields specify the default value and the environment variable of these options, which are typically set in `.cdnsenv`. The default value is `5000`. Environment variable: `pnrMaxPort`. For information on the corresponding parameters for the `ihdl_parameter` file, see Verilog In Command-Line Mode.

Field	Description
Pin Placement	Specifies the direction for placing pins in the schematic that Verilog In generates using the pin-placement options.

Placement Configuration	Specifies the configuration to be used for pin-placement in the schematic.
	■ *Left and Right Side*
	Places all the 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*
	Places the pins on any side.
	■ *Pin Placement File*
	Enables *Pin Placement File Name* field.
	■ *Pin Placement File Name*
	Specifies 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`
	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}`

Field	Description
	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` Based on the format of the entries in the pin-placement file, the following behavior of the pin-placement functionality is observed: **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.

Field	Description
Reference Schematic/ Symbol View for Inherited Connections	This section lets you specify a list of views that can be looked up by the tool.
List of Views	Specifies the 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.

Field	Description
Through Cellview to be Used for Port Shorts	This section specifies the library, cell, and view name of the component to be used between shorted ports.
Library	Lets you specify the library 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`. 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.
Cell	Lets you specify the cell name of the component to be used between shorted ports. The default location of this symbol is `cds_thru`.
View	Lets you specify the view name of the component to be used between shorted ports. The default location of this symbol is `symbol`.
Continuous Assignment Symbol	Specifies the library, cell, and view names for the patch symbol to 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.
Library	Lets you specify the library name. The default location of this symbol is the `basic` library.
Cell	Lets you specify the cell name. The default location of this symbol is the `patch` library.
View	Lets you specify the view name. The default location of this symbol is the `symbol` library.

Field	Description
Advanced Schematic Generation Options	This section lets you set advanced schematic generation options.
Generate Square Schematics	Specifies whether to square the schematic. Deselect this option if you do not want to have Verilog In manipulate rows and columns of devices to make a rectangular schematic into a square one. By default, this option is `enabled`.
Minimize Crossovers	Specifies whether to minimize crossover of nets. Select this option to minimize crossovers of nets. By default, this option is `disabled`.
Optimize Wire Label Locations	Specifies whether to override default label placement. Select this option 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. By default, this option is `disabled`.
Extract Schematics	Specifies whether to extract the schematic. Select this option 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. By default, this option is `enabled`.

Field	Description

Ignore Extra Pins on Symbol	Lets you control the selection of symbols from the reference libraries. If this button is selected and Verilog In finds a reference symbol with the same name as specified in the Verilog design, the symbol is picked up. The pins not referred remain unconnected in the schematic.
	
	By default, this option is `disabled`.
	
	The symbol is 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 `pin_master_cells` 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 `pin_master_basic_lib` specifies whether the symbol views of pins must 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	Specifies whether to connect dummy nets to unconnected pins of instances. Deselect this option to connect dummy nets named `_NeTt_1,2,3...` in the netlist view to unconnected pins of instances. By default, this option is `disabled`. This option only works for netlist view and not for schematic view. For schematic view, all unconnected pins on instances are always connected to dummy nets named `NeTt_1,2,3...`.
Verbose	Lets you print detailed status messages while the schematic is being partitioned and routed. Select this option to print detailed messages. By default, this option is disabled.

Field	Description
Create Snap Space Properties	Enables or disables 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	Lets you specify 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	Specifies 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. 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. For more details on the Text-to-Symbol generator and the `tsg` files, see Text-to-Symbol Generator.

Field	Description
Schematic Dimensions	This section provides options to let you set schematic dimensions.
Sheet Symbol	Specifies 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 Studio Design Environment. A selection of sample sheet symbols is shipped with Virtuoso Studio 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	Specifies 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 fit on each sheet up to the maximum of 1024. The default is 1024.

Field	Description
Maximum Number of Columns	Specifies 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 fit on each sheet up to the maximum of `1024`.
	The default is `1024`.
Font Height	Controls 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	Specifies 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 `0.125` to `0.74` inches.
	The default is `0.2`.
Line To Line Component Spacing	Specifies 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 `0.125` to `0.74` inches.
	The default is `0.5`.
Component Density	Lets you 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`.

Related Topics

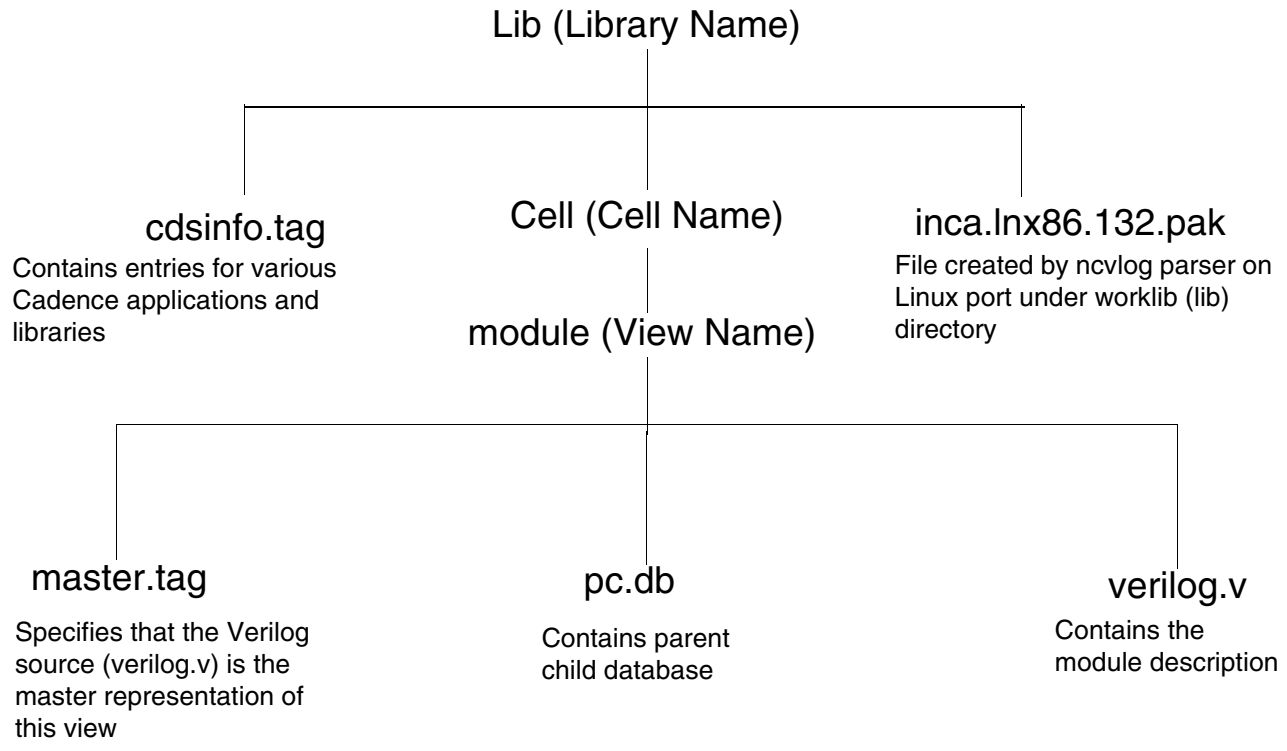
Global Net Options

Import Options

B

Pre-Compiled Libraries in Verilog In

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.

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 must 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 can 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. You can 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.

   Consider the following example where *Lib* is the logical library name:

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

   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.

Uses of 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 as follows:

- 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 must 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 files from which it has been created is modified.

Limitations of Using Pre-Compiled Libraries in Verilog In

The limitations in using pre-compiled libraries are as follows:

- 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.

Acceleration of Pre-Compiled Library Creation

It is recommended that you always use Pre-Compiled librarylibrary if they are available or if the same library are being used in multiple runs. You must create the pre-compiled library if they are not available. If you do not have any pre-compiled library and if, only a few modules are being referenced from the library, then you must use them as normal library. If the number of modules being referenced is large, you can specify a target IR library which must preferably be in the `/tmp` area or on a fast access disk area. This reduces the design 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.

Ensure that you have enough swap or disk space for the target IR library.

You can accelerate the creation of pre-compiled library in case of netlist as well as schematic views. In case of netlist views, the flatter the design the better is the acceleration. In case of schematic views, you can use the fast label option if label positioning is not important.

Results of Accelerating the Design Import Process

Speed for design import for the netlist view has been increased.

Example 1: A sample system specification for which the speedup results were observed are as follows:

```
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 acceleration results for both the original and new time without pre-compiled libraries for netlist view are as follows:

No of Modules	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

Example 2: Another sample system specification for which the speedup results were observed are as follows:

```
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 acceleration results for both the original and new time with pre-compiled libraries for netlist view are as follows:

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