cādence®

Virtuoso NC-Verilog Environment User Guide

Product Version IC23.1
September 2023

Contents

1 About the NC-Verilog Integration Environment

Virtuoso® Verilog Environment for NC-Verilog Integration (NC-Verilog Integration Environment) lets you netlist digital designs. This environment also integrates with other Cadence tools to simulate, analyze, and debug designs.

The NC-Verilog Integration Environment provides a methodology to netlist and simulate digital designs. This methodology includes the following stages:

1. Initialize the run directory for setting the environment.
2. Generate the netlist of the design that describes the connectivity of the design.
3. Simulate the design in interactive or batch mode using the generated netlist as an input.
4. Compare simulation databases.

This user guide explains how you use the NC-Verilog Integration Environment. It is aimed at 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 Layout Suite and Virtuoso Schematic Editor.
- The design and use of various types of parameterized cells.
- Verilog® Hardware Description Language, which you use to develop models that describe your design and its environment.
- Component Description Format (CDF), which enables you to create and describe customized components in the Virtuoso Studio design environment.
- The Cadence Hierarchy Editor software, which is used to view multiple levels of a single design and create new configurations that provides expansion information for mixed-signal partitioning.

This topic describes the following:

- NC-Verilog Integration Tools on page 11
- Entering the Environment on page 12
- Main NC-Verilog Window on page 17
- SimVision Window on page 17
- SimCompare on page 19
- Managing the Run Directory on page 20
    - Test Fixture Files on page 20
- Using Command Line Interface on page 20

Licensing Requirements

You need to have the Virtuoso Schematic Editor Verilog® Interfaces (license number 21400) license to use NC-Verilog Integration Environment.

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

NC-Verilog Integration Tools

The NC-Verilog Integration Environment consists of the following:

- The Virtuoso® Verilog Environment for the NC-Verilog Integration window (main NC-Verilog window)

  The main NC-Verilog window appears when you first access the NC-Verilog Integration Environment. The main NC-Verilog window provides access to simulation commands, command forms, and support tools.

- The SimVision window

  The SimVision window, which you launch from the main NC-Verilog window, is used to interactively simulate and debug the design. You use the SimVision window to directly interact with the simulator. You can open a database, trace signals, set breakpoints, observe signals, and perform many other functions to verify your design.

- The SimCompare tool

  The SimCompare tool is used by NC-Verilog in background to compare the results obtained from different simulations. It is invoked when you select Simulation Compare option from the Fixed Menu. SimCompare provides a text description of any differences found.

Entering the Environment

The NC-Verilog Integration Environment for simulating and debugging mixed-level logic designs integrates with Virtuoso Schematic Editor.

The procedures described in this section show you how to enter the NC-Verilog Integration Environment and how to access the main NC-Verilog window. This main window provides access to the commands, forms, and tools you use to simulate and debug mixed-level designs.

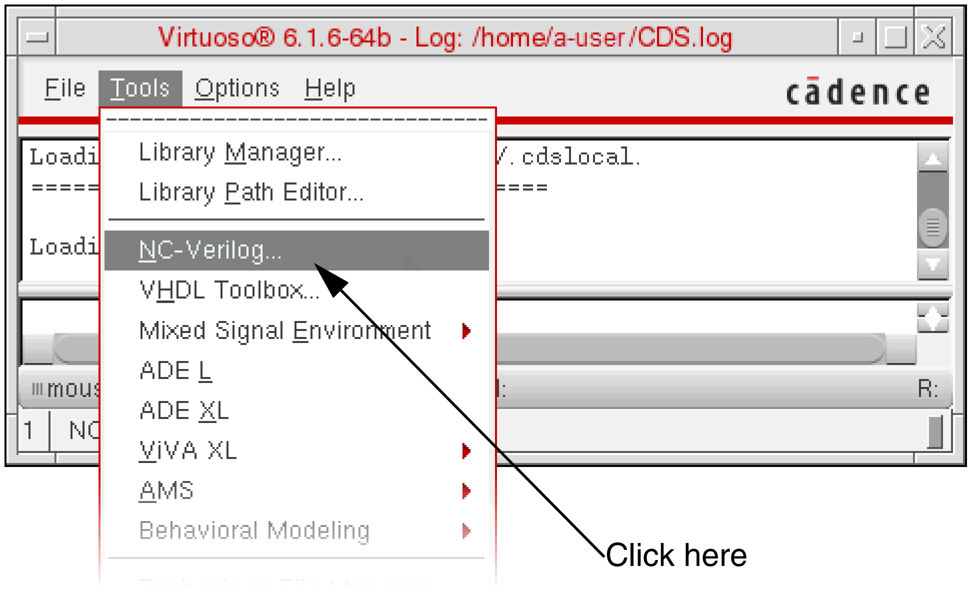
You can enter the NC-Verilog Integration Environment from the following:

- Virtuoso Command Interpreter Window (CIW)
- Virtuoso Schematic Editor

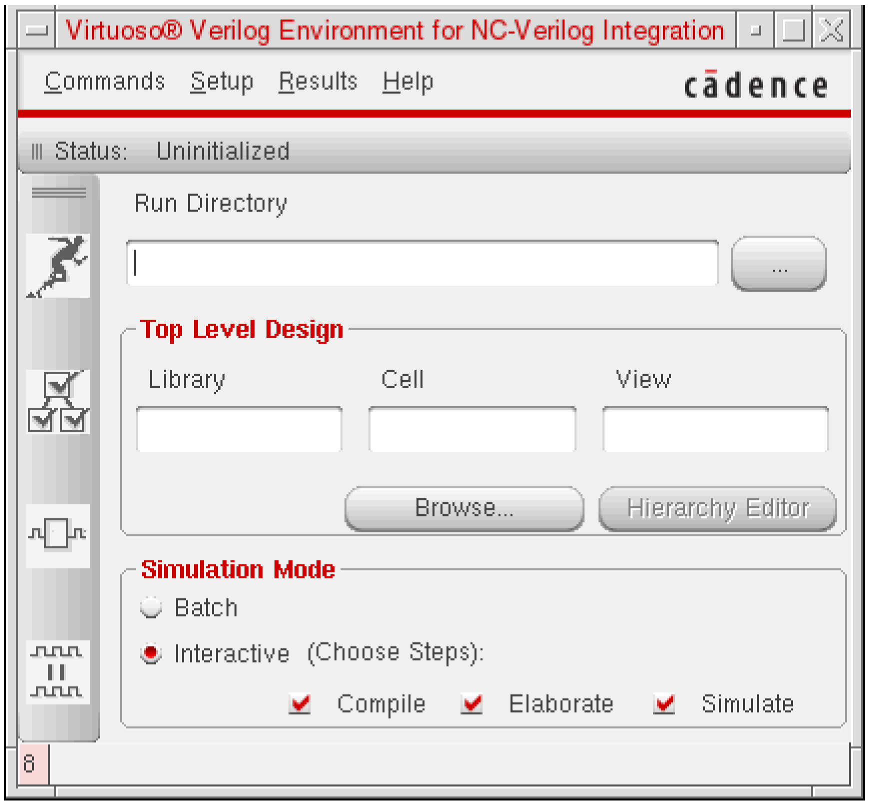
Entering the Environment from CIW

Use the following procedure to enter the NC-Verilog Integration Environment from the CIW:

- Choose *Tools – NC-Verilog*.



The Virtuoso® Verilog Environment for NC-Verilog Integration window appears.


Virtuoso® Verilog Environment for NC-Verilog Integration
Commands Setup Results Help
cādence
...
Browse...
Hierarchy Editor
8

Status: Uninitialized

Run Directory

Top Level Design

Library Cell View

Simulation Mode

Batch

[✓ Interactive] (Choose Steps):

✓ Compile ✓ Elaborate ✓ Simulate

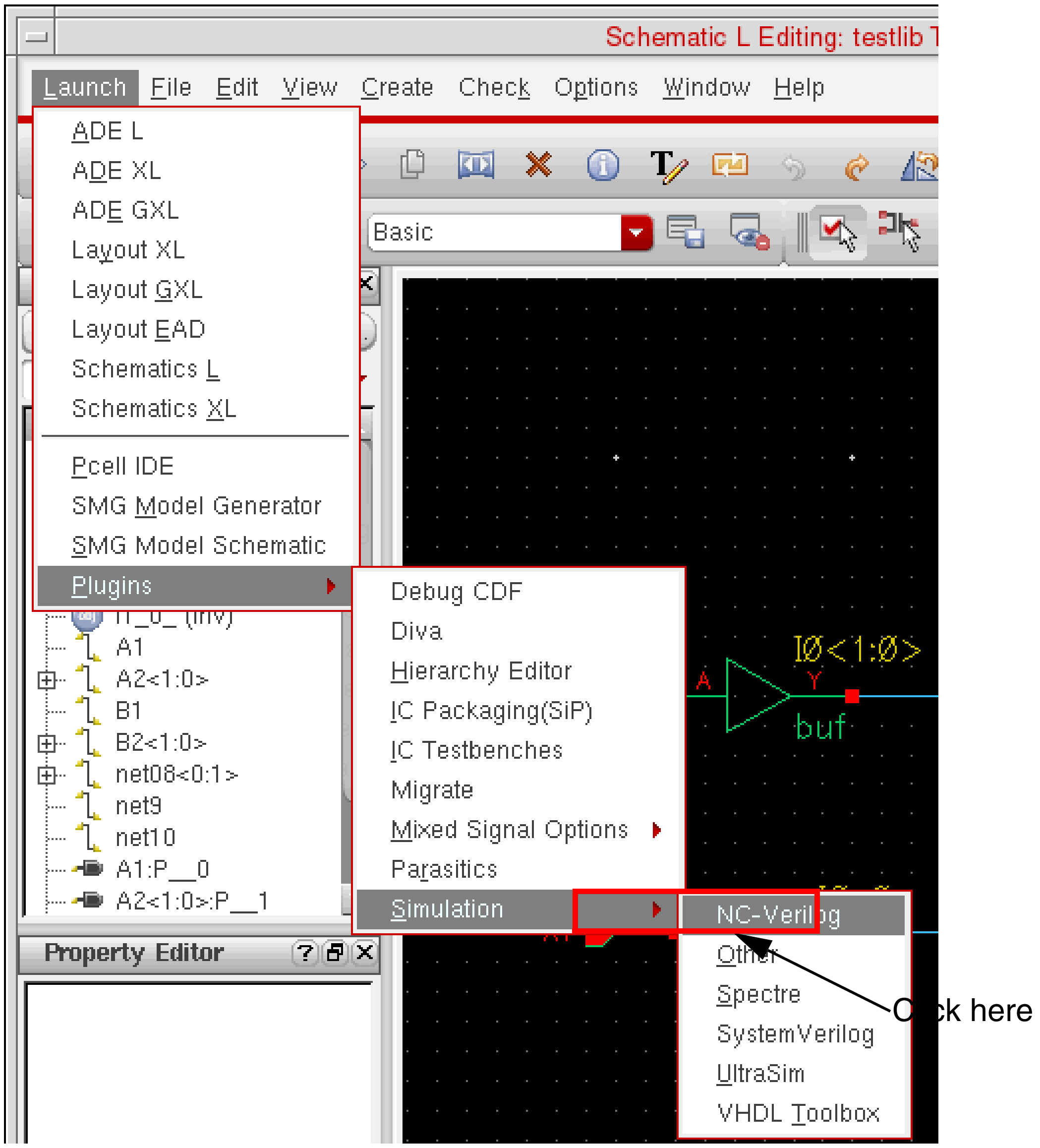
Note: When you first enter the NC-Verilog Integration Environment from the CIW, the fields in the window are blank. When you restart NC-Verilog Integration from the CIW, the fields contain information about the design last simulated.

You now have access to the commands, forms, and tools that you need to simulate your design.

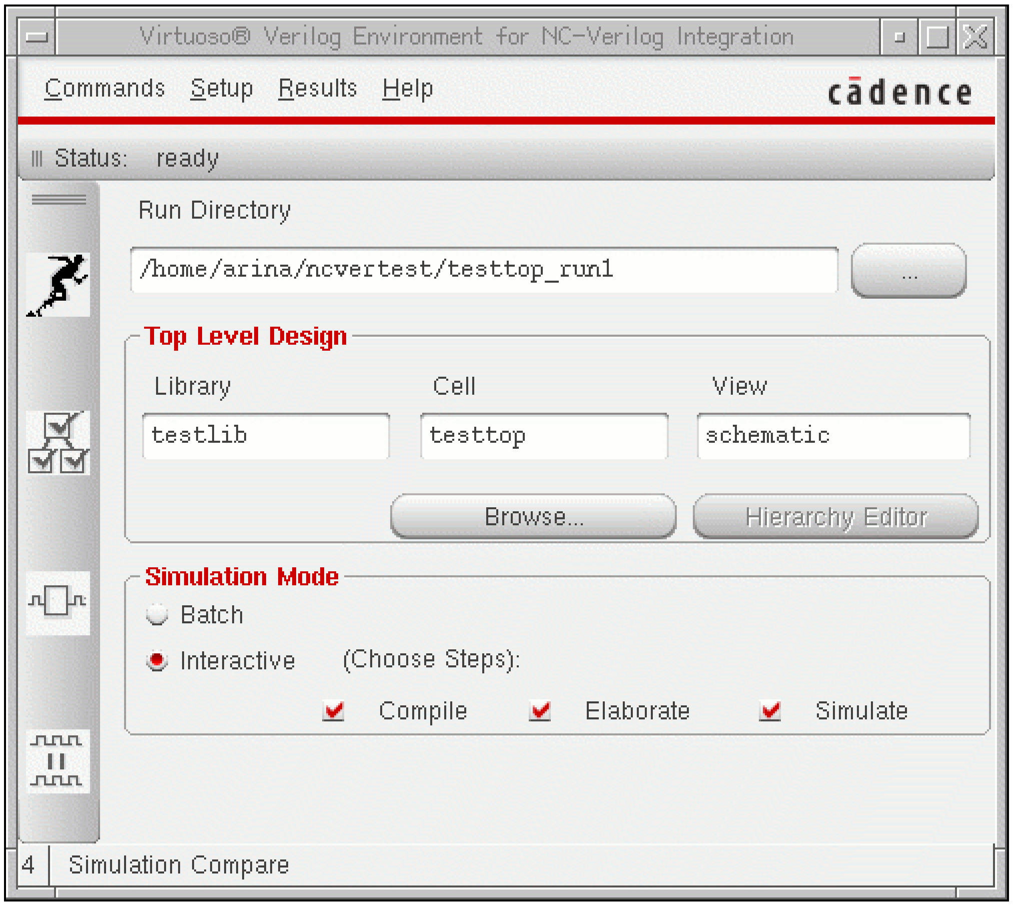
Entering the Environment from a Schematic View

Use the following procedure to enter the NC-Verilog Integration Environment from a schematic editor window.

- Choose *Launch – Plugins – Simulation – NC-Verilog*.



The Virtuoso® Verilog Environment for NC-Verilog Integration window appears.



Run Directory: /home/arina/ncvertest/testtop_run1

Top Level Design: Library testlib, Cell testtop, View schematic

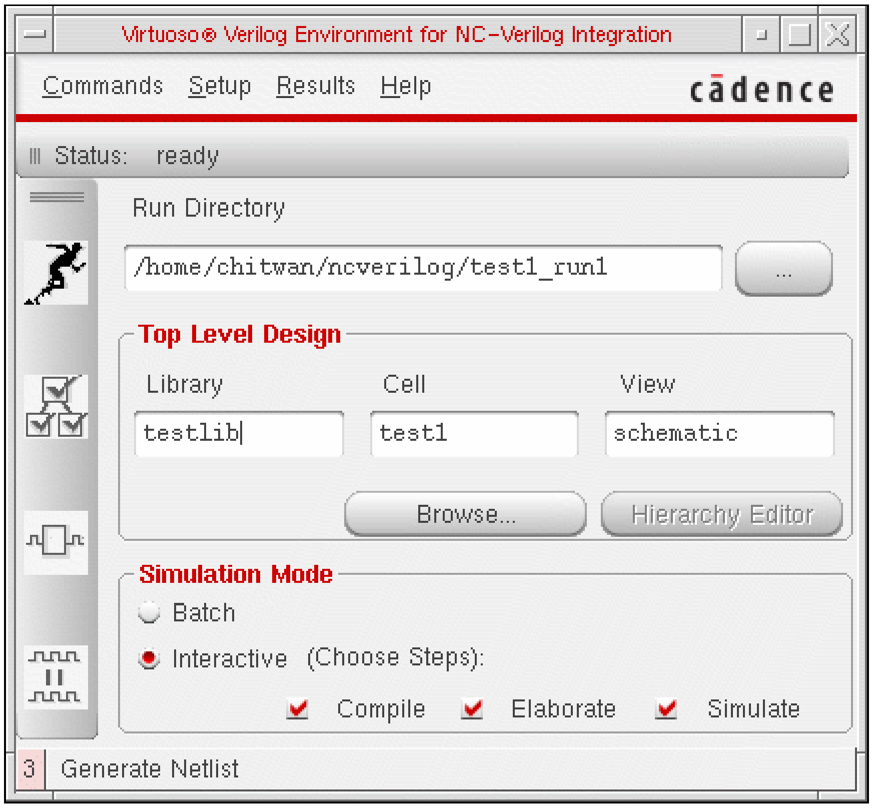
Simulation Mode: ( ) Batch [✓ Interactive] (Choose Steps): ✓ Compile ✓ Elaborate ✓ Simulate

Note: The system automatically initializes the run directory name and design information derived from the schematic.

You now have access to the commands, forms, and tools that you need to simulate your design.

Main NC-Verilog Window

The Virtuoso® Verilog Environment for NC-Verilog Integration window appears when you first access the NC-Verilog Integration Environment. The main NC-Verilog window provides access to simulation commands, command forms, and support tools. The following figure shows the main NC-Verilog window:


Virtuoso® Verilog Environment for NC-Verilog Integration

Commands Setup Results Help

Status: ready

Run Directory: /home/chitwan/ncverilog/test1_run1

Top Level Design

Library	Cell	View
testlib	test1	schematic

Browse... Hierarchy Editor

Simulation Mode
Batch [✓ Interactive] (Choose Steps):
✓ Compile ✓ Elaborate ✓ Simulate

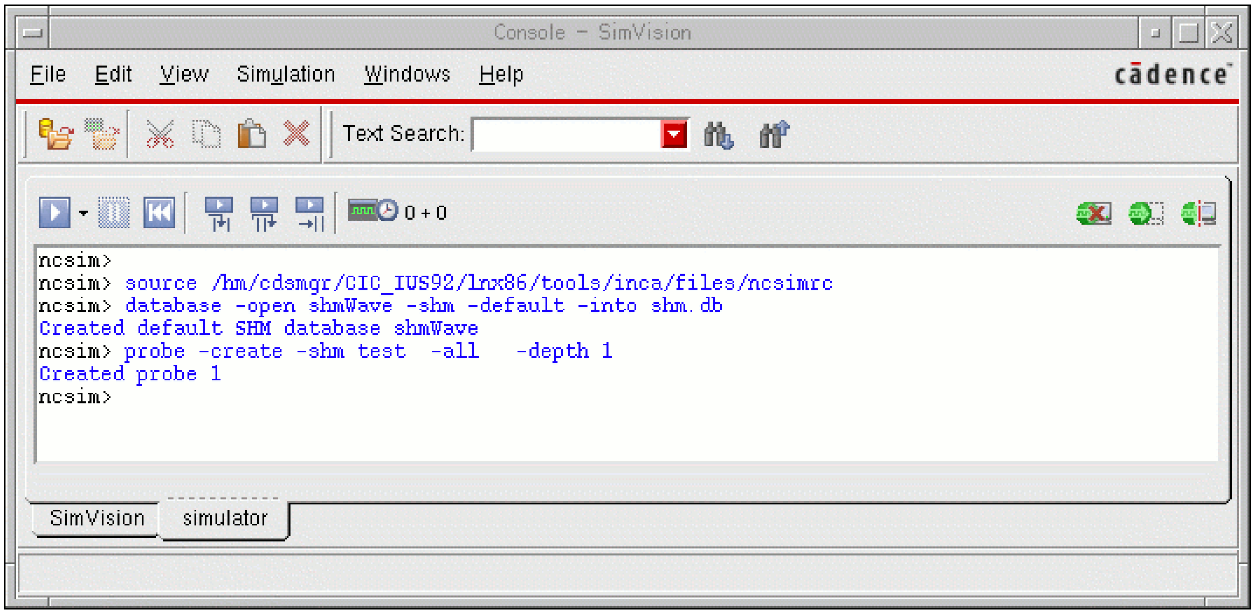
3 Generate Netlist

The main NC-Verilog window sections include the Status line, Menu banner, Fixed menu, Run Directory field, Top Level Design selection fields, library browsing button, Hierarchy Editing button and Simulate Options buttons.

SimVision Window

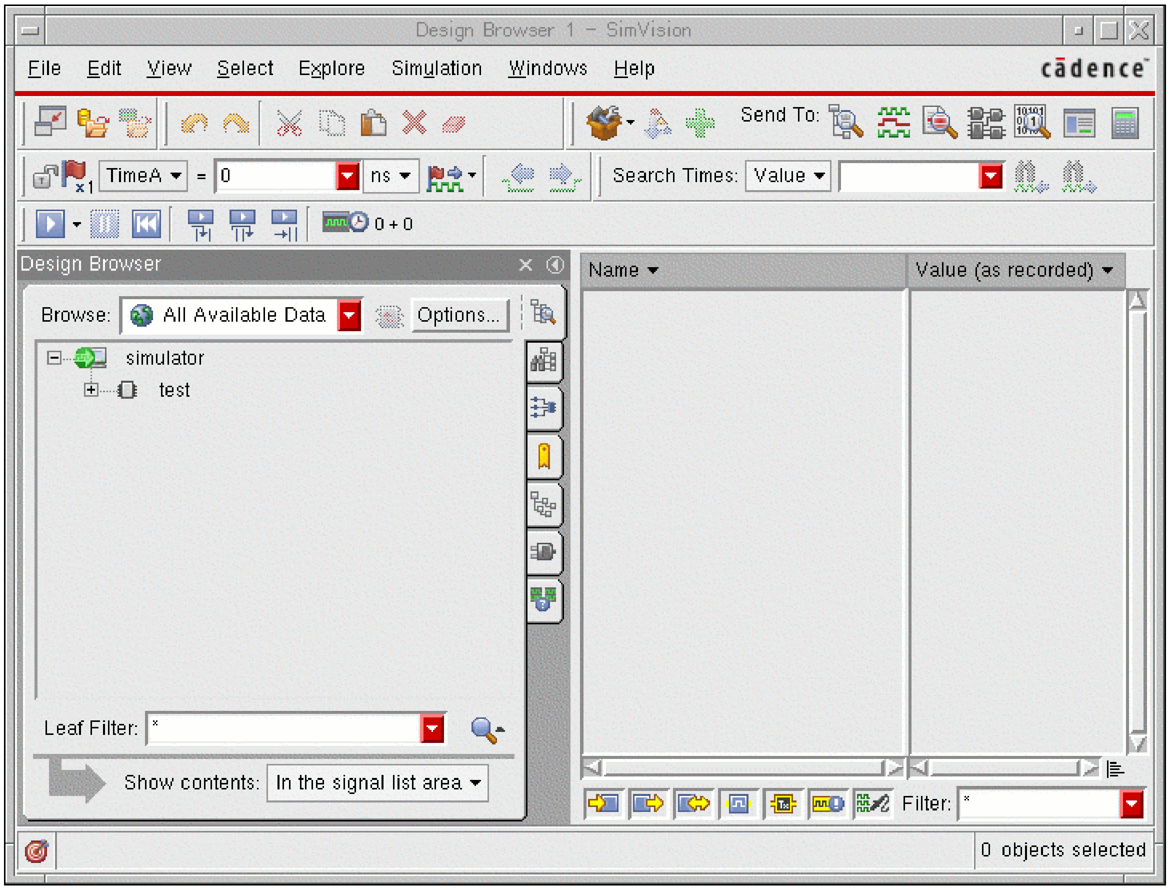
The SimVision window is invoked when you perform an interactive simulation. In an interactive simulation, you can select the option to compile, elaborate and simulate the design from the main NC-Verilog window. When you select all the three options, the system invokes the *Console - SimVision* and *Design Browser SimVision* windows upon successfully compiling and elaborating the design.

Console - SimVision



```
ncsim>
ncsim> source /hm/cdsmgr/CIC_IUS92/lnx86/tools/inca/files/ncsimrc
ncsim> database -open shmWave -shm -default -into shm.db
Created default SHM database shmWave
ncsim> probe -create -shm test  -all   -depth 1
Created probe 1
ncsim>
```

Design Browser - SimVision



The Simcontrol tool is part of the Cadence® SimVision Analysis environment which is a unified graphical debug environment for Cadence simulators. For more information on the SimControl user interface, refer to the Cadence® *SimVision Analysis Environment User Guide*.

SimCompare

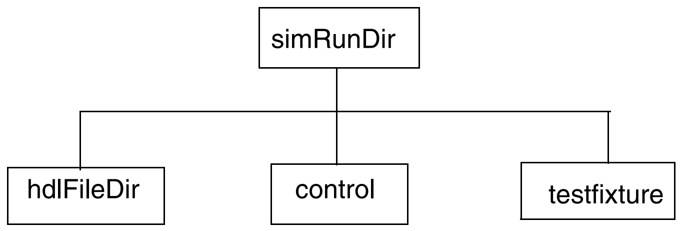
SimCompare is a comprehensive tool for comparing simulation results. NC-Verilog makes use of this tool to perform comparison of simulation results. It is invoked in the background when you select the Simulation Compare option from the Fixed Menu. You can also invoke the SimCompare tool from the Commands Menu in the main NC-Verilog window.

SimCompare provides a text view of any differences found when the simulation results are compared. It accepts databases in either SST2 or VCD format. For more information on using the SimCompare tool, refer to the *SimCompare User Guide*.

Note: If SimCompare is not available, NC-Verilog makes use of the Comparescan tool for performing simulation comparison. It is flexible, programmable, and allows a large number of applications. For more information on using the Comparescan tool, refer to the *Comparescan User Guide*.

Managing the Run Directory

The run directory is the directory in which you and the system create and store files during the simulation process. For example, the run directory is where you create the stimulus (test fixture files) and where the system stores the simulation results.



Test Fixture Files

In the NC-Verilog Integration Environment, a test fixture provides the stimulus that drives the simulation.

A test fixture can be either

- The stimulus only
- A module that contains the stimulus

Refer to Chapter 6, “Working with the Stimulus”for detailed information on test fixtures.

Using Command Line Interface

In NC-Verilog Integration Environment, you can generate a netlist in a standalone mode using command line options. To generate a netlist, you should do the following:

- Create the `si.env` file in the run directory and add verilog netlister variables to the file. For information on the variables that the Verilog netlister uses while netlisting a design, refer to Netlist Setup Form on page 195.

  Note: For more information on netlister variables, refer to the SE Variables section in

Chapter 3, Customizing the Simulation Environment (SW) of Open Simulation System Reference.

- Run the following si command, which represents an OSS binary file, to generate a netlist:

  ```
  si <runDirname> -batch -command netlist -cdslib cds.lib
  ```

  The command line options used with this command are:

  - `si`: Represents the OSS binary file.
  - `runDirname`: name of the run directory.
  - `-batch`: Refers to the batch simulation mode.
  - `-command`: Specifies the command to generate a netlist.
  - `-loadLocal`: Loads the `si.local` file from the simulation run directory.
  - `-cdslib`: Specifies the cds.lib library, which defines the design libraries and the path to these libraries. You are required to specify the complete path of the cds.lib library while executing this command.

Queuing Verilog Netlist Processes

When you use the command line interface, you can configure the environment to queue Verilog netlist processes when all the available licenses are being used. You use a license checking utility to queue Verilog netlist processes.

To enable the license checking utility:

➡ Set the environment variable `CDS_LIC_QUEUE_ENABLE` to `1`.

To enable the license checking utility when licenses are on multiple servers:

➡ Set the environment variable `CDS_LIC_QUEUE_POLL` to `1`.

The license checking utility waits for a minimum of ten minutes before checking the availability of licenses for queued processes. The process remains in queue only during the wait period. You can increase this wait period.

To increase the wait period of the license checking utility:

➡ Set the environment variable `CDS_LIC_QUEUE_MAXTIME_INT` to the wait period. For example, to increase the wait period to 15 minutes, set the environment variable to 15.

```
setenv CDS_LIC_QUEUE_MAXTIME_INT 15
```

Note: You can only increase the wait period from the default ten minutes. If you set `CDS_LIC_QUEUE_MAXTIME_INT` to a value less than ten, the license checking utility still waits for ten minutes, and not to the set value.

The license checking utility displays the following message every minute, till a license becomes available or the wait period is complete.

```
Waiting for license feature 21400...
```

The utility displays the following message when the wait period is over and if during this period no license became available to a process. After the wait period, the process is no longer in the queue.

```
Max Queue wait time reached for license feature 21400
```

2

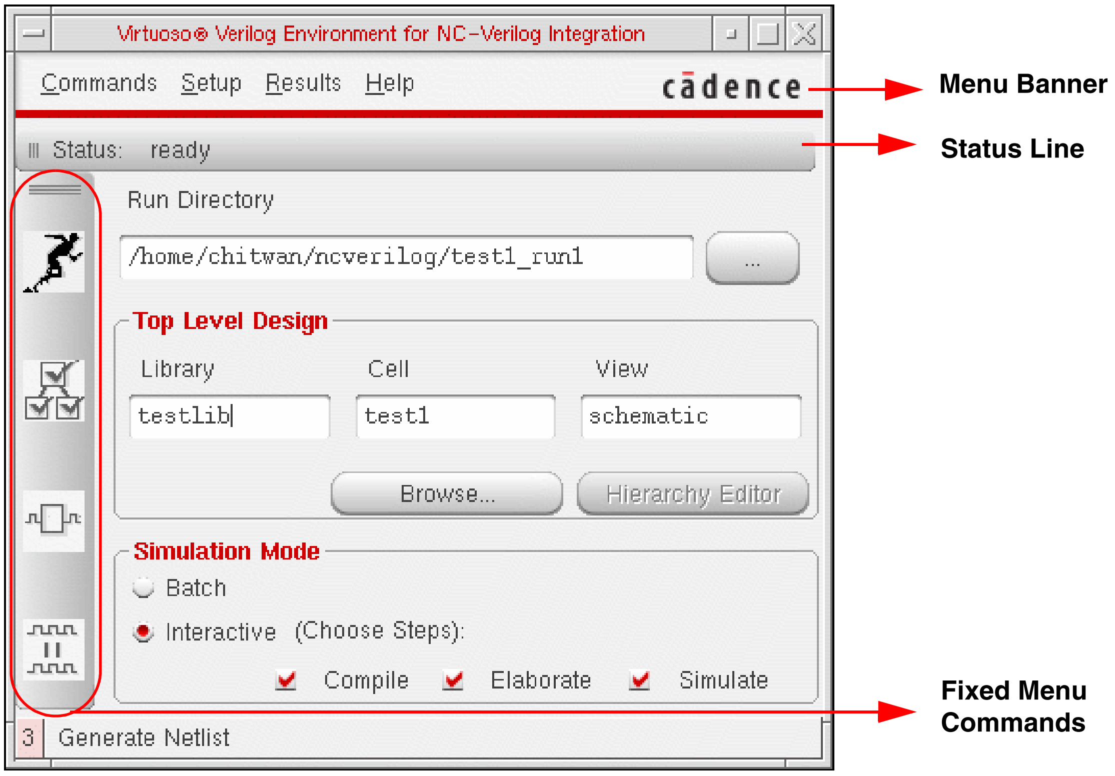
NC-Verilog Integration Control Commands

This chapter describes the commands and forms you can use from the Virtuoso® Verilog Environment for the NC-Verilog Integration window.

About the Main NC-Verilog Window

The Virtuoso® Verilog Environment for the NC-Verilog Integration window (main NC-Verilog window) gets displayed when you first enter the NC-Verilog Integration Environment. The main NC-Verilog window provides access to simulation commands, command forms, and support tools.

The following figure shows the main NC-Verilog window:



Status: ready

Run Directory: /home/chitwan/ncverilog/test1_run1

Top Level Design

Library	Cell	View
testlib	test1	schematic

Simulation Mode
- Batch
- ● Interactive (Choose Steps):

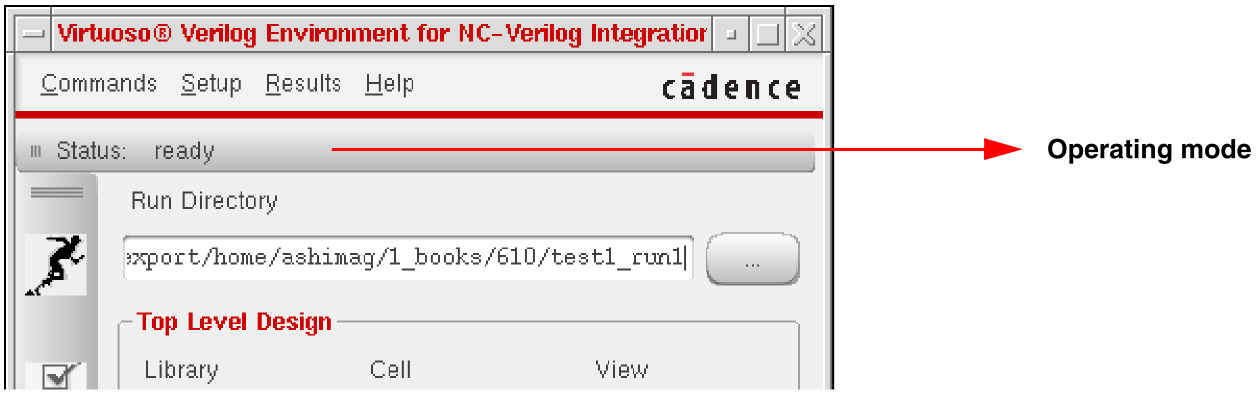
✓ Compile ✓ Elaborate ✓ Simulate

3 Generate Netlist

The main NC-Verilog window sections include the status line, menu banner, fixed menu, Run Directory field, Top Level Design selection fields and Simulate Options fields.

Status Line

The Status Line shows the current operating mode of the current design. The following figure shows the Status Line:



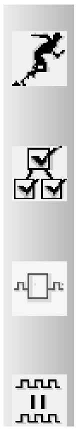
Menu Banner

The Menu Banner lets you access all the commands, forms, and tools required for controlling the simulation with the help of the following set of options:

- Commands Menu
- Setup Menu
- Results Menu
- Help Menu

Fixed Menu

The Fixed Menu provides access to the frequently used commands. For more information on its options, refer to Fixed Menu on page 62.



Run Directory

The Run Directory field is used to specify the name of the directory for design netlisting, simulation and waveform database creation. You can specify a new or already existing directory name in this field. For more information, refer to Initializing a Design.

Top Level Design Options

Use the Top Level Design options to specify the *Library*, *Cell* and *View* name of the top level design.

- Browse is used to find and open libraries, cells, and cellviews.
- Hierarchy Editor is used to display the Hierarchy Editor to edit the configuration and define the design hierarchy for netlisting and simulation. This option can only be used if the top level cellview is a 5.X configuration.

Note: Once you specify any run directory name or top level library, cell or view name you need to press the tab key for the system to accept that value and move to the next field. Alternatively you can also click the next field after you have entered the value for one of the fields.

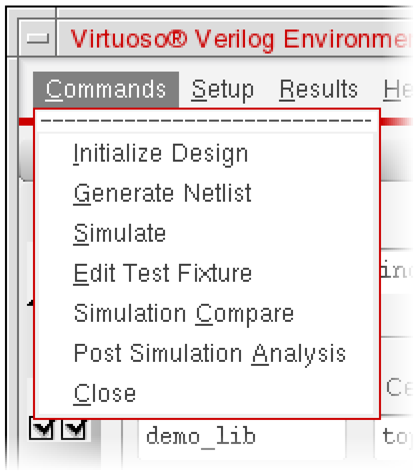
Simulate Options

Simulator Mode specifies the kind of simulation that you require. It can be either Interactive or Batch.

- In Interactive simulation you can select the steps to Compile, Elaborate and Simulate. You may select to omit performing any one of these. In this mode, the system invokes the SimControl user interface after it successfully compiles and elaborates the design. You can then interactively simulate and debug the design using the SimControl user interface.
- Batch simulation results in the execution of all the three options: Compile, Elaborate and Simulate in one go. In batch mode of simulation, SimControl user interface is not invoked.

Note: The executable and log file names will depend on the simulator being used. For the changes in the executable and log file name when using the Xcelium simulator, see Running Simulations with Xcelium on page 211.

Commands Menu



- *Initialize Design*

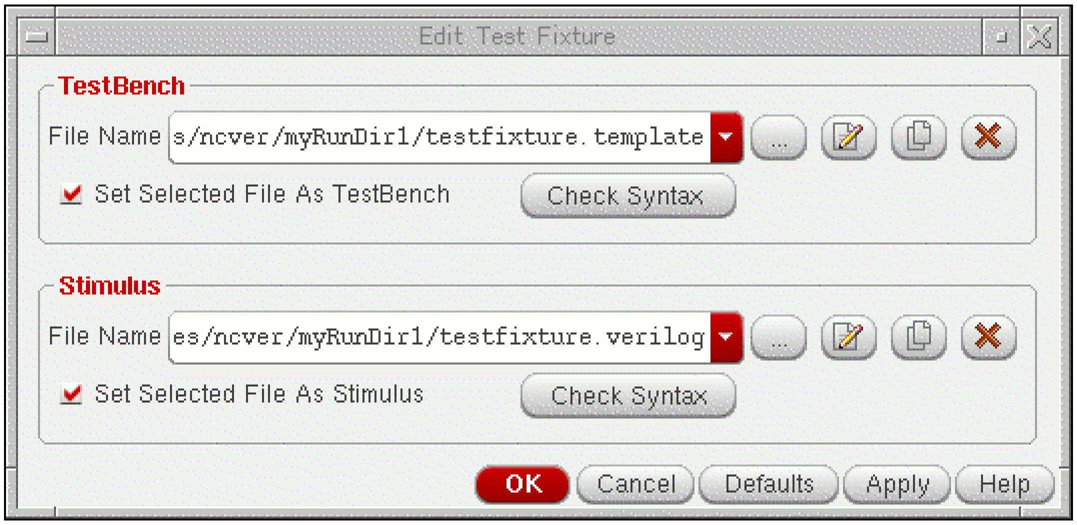
  Starts the NC-Verilog simulator. You must initialize design before you can access the other fixed menu commands.
- *Generate Netlist*

  Triggers the OSS netlist() routine to generate an incremental hierarchical netlist in the specified run directory.
- *Simulate*

  Simulates the design in either interactive or batch mode.

- *Edit Test Fixture*

  Displays the *Edit Test Fixture* dialog box. This is used to specify the stimulus file or the test bench for simulation.


Edit Test Fixture
TestBench
File Name s/ncver/myRunDir1/testfixture.template
✓ Set Selected File As TestBench
Check Syntax
Stimulus
File Name es/ncver/myRunDir1/testfixture.verilog
✓ Set Selected File As Stimulus
Check Syntax
OK
Cancel
Defaults
Apply
Help

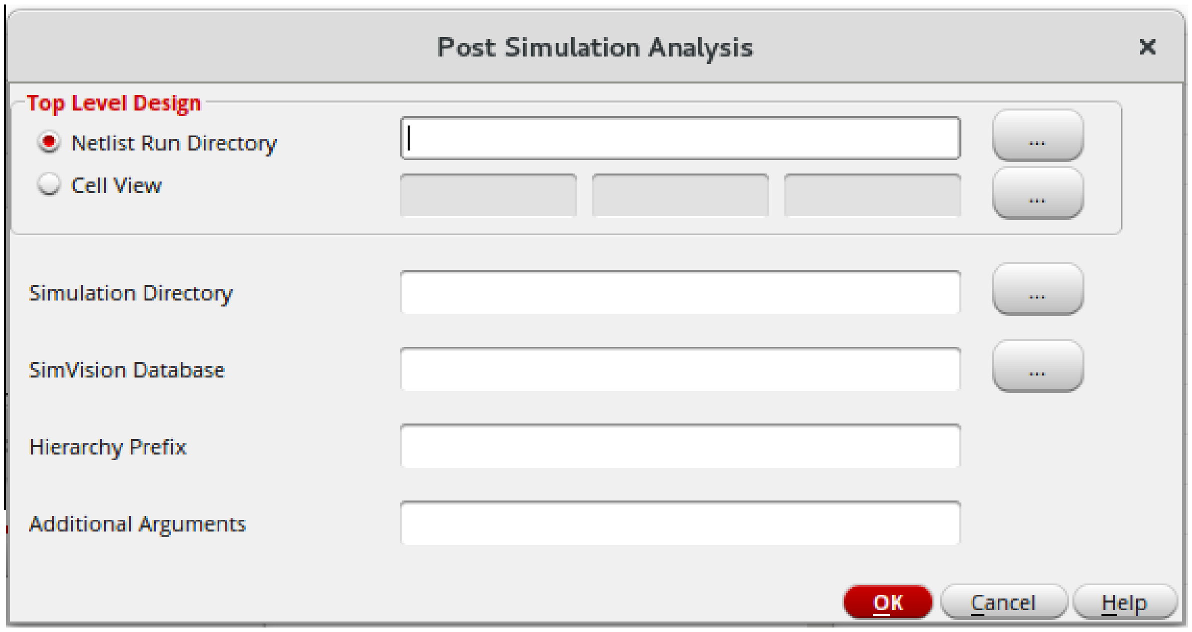
  The default stimulus file and test bench created by the netlister are `testfixture.verilog` and `testfixture.template`, respectively. You can edit these default files or copy them to create new files using the *Edit Test Fixture* form. For more information, refer to Chapter 6, "Working with the Stimulus".

- *Simulation Compare*

  Invokes the SimCompare tool to compare the golden and test SST2 databases. You need to specify these databases and other options for comparison using the Simulation Comparison Setup form.

- *Post Simulation Analysis*

  Displays the Post Simulation Analysis form. Use this form to initiate cross-selection of objects between SimVision and Virtuoso Schematic Editor using the available simulation data of a design.

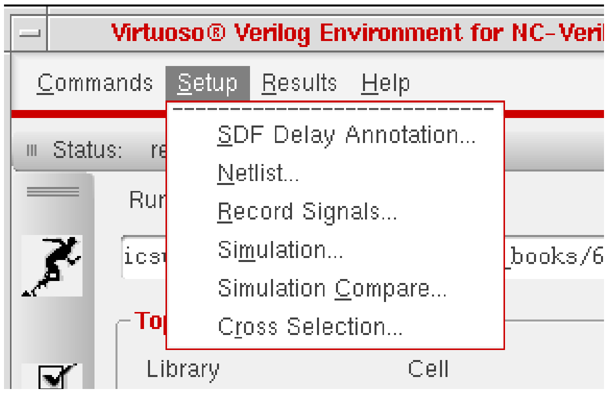
  

  For details, see Cross-Selecting Objects Using Simulation Data. For detailed information on SimVision, see the Cadence *SimVision Analysis Environment* documentation suite.

- *Close*

  Terminates the current simulation.

Setup Menu

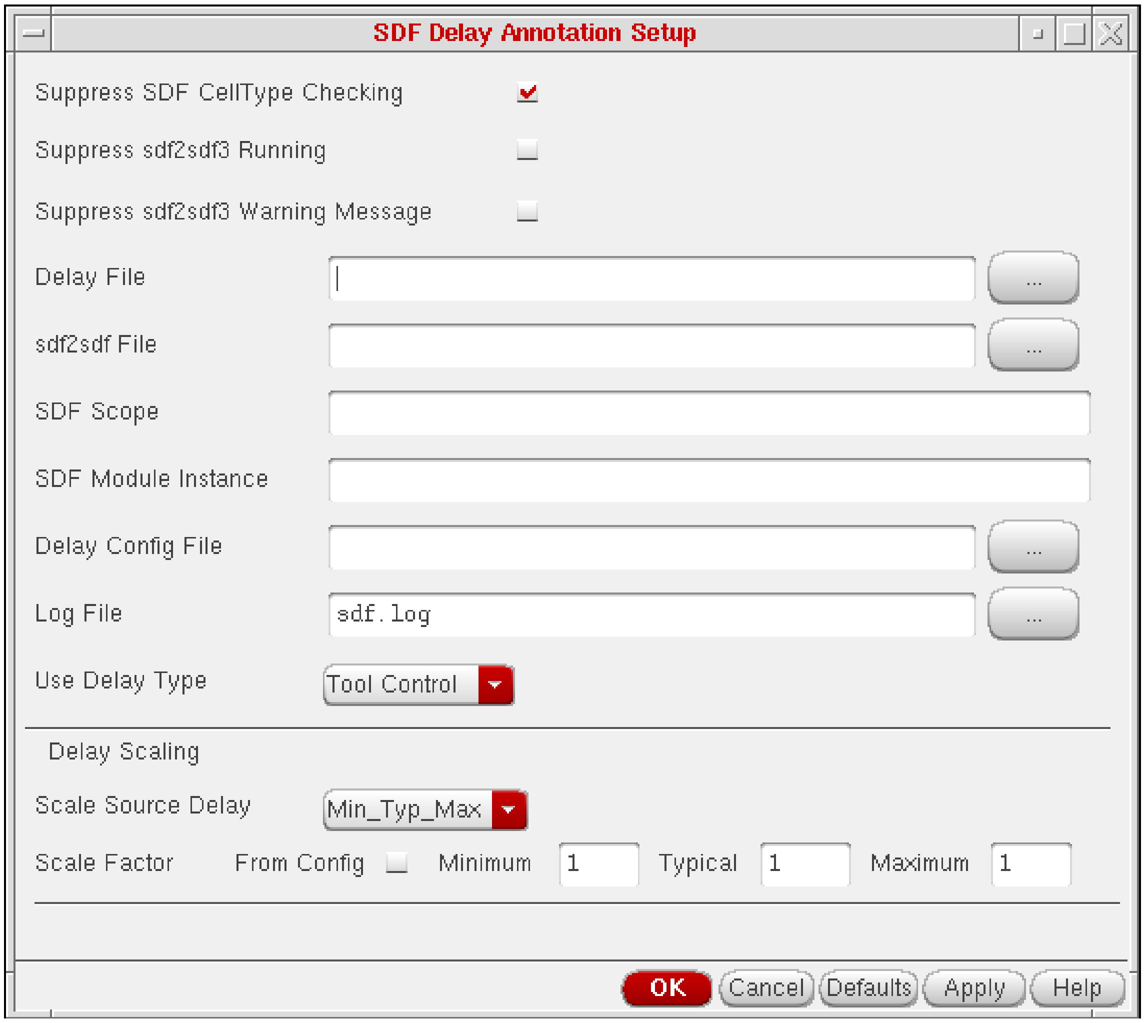


The following tables describe the below mentioned Setup menu options:

- SDF Delay Annotation
- Netlist
- Record Signals
- Simulation
- Simulation Compare
- Cross Selection Setup

SDF Delay Annotation

Selecting this option displays the SDF Delay Annotation Setup form. In this form, you can specify an SDF delay file to be prepared for ncelab to annotate. The SDF delay file will be converted from the Virtuoso Design Editor L name space to Verilog name space by running sdf2sdf3 (this is an optional feature). After this, it is compiled using ncsdfc. Next, the compiled SDF file is stored in the run directory under the name <Original SDF File Name>.compiled. Finally, an SDF command file will be created which will then be passed on to ncelab during elaboration time.


SDF Delay Annotation Setup

Suppress SDF CellType Checking	✓
Suppress sdf2sdf3 Running	
Suppress sdf2sdf3 Warning Message	
Delay File	
sdf2sdf File	
SDF Scope	
SDF Module Instance	
Delay Config File	
Log File	sdf.log
Use Delay Type	Tool Control

Delay Scaling

Scale Source Delay	Min_Typ_Max
Scale Factor	From Config ☐ Minimum 1 Typical 1 Maximum 1

Setting the Delay Annotation Options

The following table describes the various delay annotation options:

SDF Delay Annotation Options

Options	Description
Suppress SDF CellType Checking	Suppresses the CellType mismatch checking performed by NC-Verilog SDF backannotator. When enabled (default), "`+sdf_nocheck_celltype`" is sent to NC-Verilog.
Suppress sdf2sdf3 Running	Suppresses the sdf2sdf3 process when you start a simulation. When selected, the simulator reads the sdf2sdf information from the last simulation. **Note:** You must run the sdf2sdf3 process at least once to create an sdf2sdf file. When you run sdf2sdf3, you must complete the *SDF Scope* field. When the simulator does not find an sdf2sdf file, an error message appears.
Suppress sdf2sdf3 Warning Message	Suppresses warning messages generated by sdf2sdf3. When enabled, a -nowarn is sent to sdf2sdf3.
Delay File	It is an SDF file used to backannotate your design. If the file is in the current directory, you enter the filename. If the file is in a different run directory, you enter the full path.
sdf2sdf File	Stores information generated by the `sdf2sdf3` process. When you run the sdf2sdf3 process, the simulator writes information to the file specified in this field. When you suppress the sdf2sdf3 process, the simulator reads information from the file specified in this field.
SDF Scope	Specifies the scope you want the system to preappend to the sdf2sdf file. You must complete this field when you run `sdf2sdf3` process.
SDF Module Instance	Specifies the module that the SDF file annotates. You do not specify this option when backannotating an entire design.
Delay Config File	Specifies an annotator configuration file. An annotator configuration file contains commands that let you control backannotation. Refer to the SDF Annotator User Guide for more information.
Log File	Specifies a log file other than the default `sdf.log` file. The log file stores the warning, error, and informational messages generated when you run sdf2sdf3.

SDF Delay Annotation Options

Options	Description
Use Delay Type	Specifies which of the three delay types (*Minimum*, *Typical*, or *Maximum*) to use for annotation. The delay type is derived from the SDF file unless you specify another source with the *Tool Control* or *From Config* options (see below). ■ *Tool Control* takes the delay type from the Simulation Options form (Refer to the *Setup – Simulation* command). ■ *Minimum* takes the minimum delay value from the SDF file. ■ *Typical* takes the typical delay value from the SDF file. ■ *Maximum* takes the maximum delay value from the SDF file. ■ *From Config* takes the delay value from the delay type specified in the *Delay Config File* field.
Delay Scaling	Enables you to factor the value of delay types. To apply these options to your simulation, you must select a delay type from the *Use Delay Type* field described above.
Scale Source Delay	It is the delay type whose value is factored by the *Scale Factor* options. ■ *Min_Typ_Max* takes the minimum, typical, and maximum values from the SDF file. ■ *Minimum* takes the minimum delay value from the SDF file. ■ *Typical* takes the typical delay value from the SDF file. ■ *Maximum* takes the maximum delay value from the SDF file. ■ *From Config* takes the delay type from the Delay Configuration file.

SDF Delay Annotation Options

Options	Description
Scale Factor	Specifies multipliers for the *Scale Source Delay* value. You enter real number multipliers in the *Minimum*, *Typical*, and *Maximum* fields. ■ *From Config* applies scale factors supplied in the Delay Configuration file. ■ *Minimum* is the factor by which NC-Verilog scales the minimum value for each delay. ■ *Typical* is the factor by which NC-Verilog scales the typical value for each delay. ■ *Maximum* is the factor by which NC-Verilog scales the maximum value for each delay. **Note:** You can enter scaling instructions in the annotator configuration file that you specify in the *Delay Config File* field described above.

Note: The *Use Delay Type*, *Scale Source Delay*, and *Scale Factor* options are used to determine the delays annotated to the Verilog design for simulation. See the example that follows.

Example of Determining Delays

The following sample shows an SDF delay entry for a timing path from A to Z.

```
SDF Delay File: (IOPATH A Z (2:3:5))
```

The scaling example is based on the following delay annotation settings:

```
Scale Source Delay:    Maximum
Scale Factor:       Minimum = .5, Typical = 1,    Maximum = 2
Use Delay Type:    Minimum
```

The simulator uses the following process to scale the delays and to determine which single delay is annotated for simulation:

1. The *Scale Source Delay* setting specifies to use the maximum delay in the SDF file as the initial value for scaling *Minimum*, *Typical*, and *Maximum* delay values.

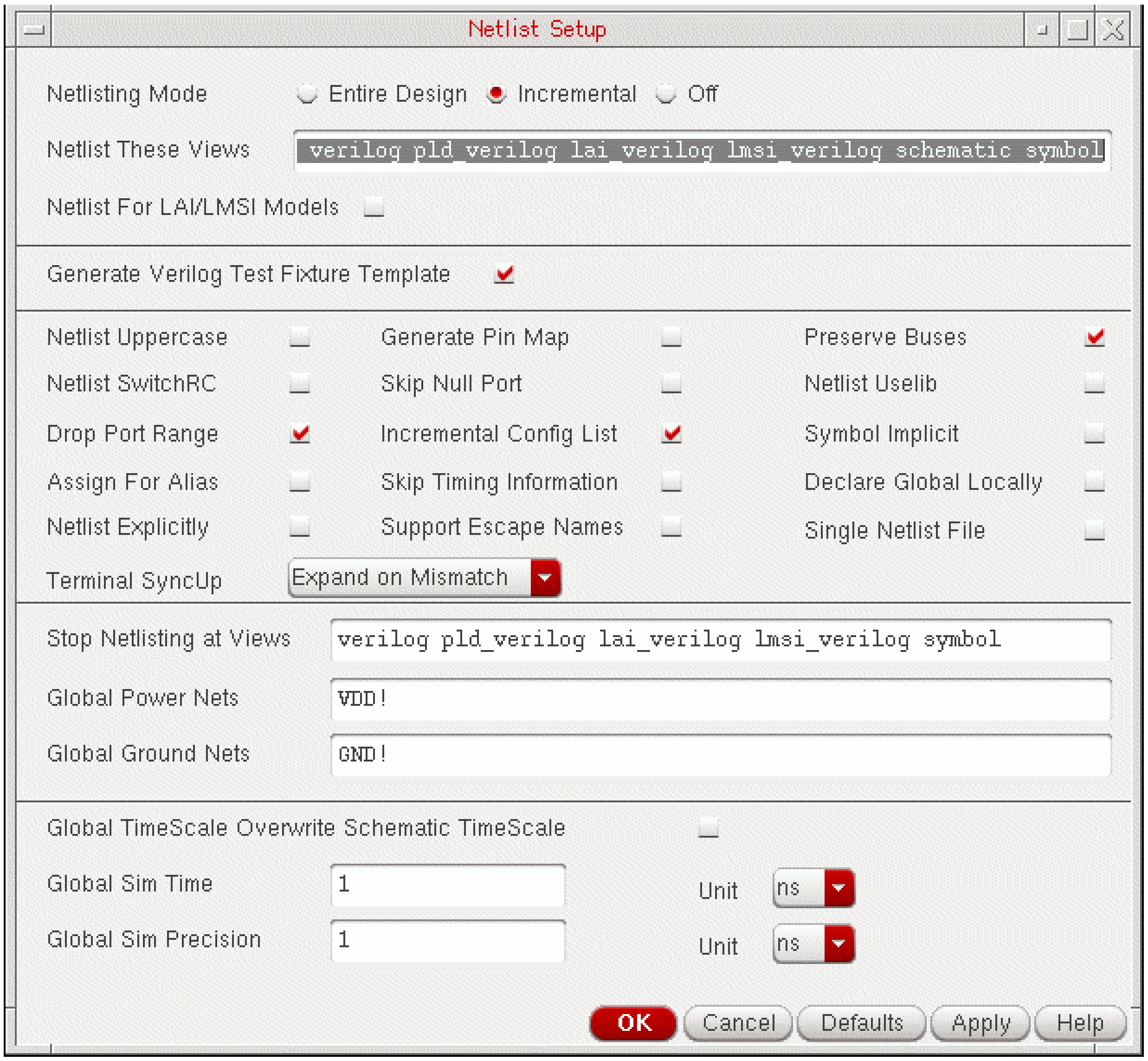
2. The *initial value* multiplied by the *factor value* results in the *scaled delay value,* as shown in the following table.

		Minimum	**Typical**	**Maximum**
(Multiplicand)	Initial Value	5	5	5
(Multiplier)	Factor Value	0.5	1	2
(Product)	Scaled Delay Value	2.5	5	10

3. The *Use Delay Type* setting specifies which scaled delay to annotate to the simulation. In the sample setting, *Minimum* is specified.
4. The value of 2.5 is annotated to the simulation.

Netlist

Selecting this option displays the Netlist Setup form. You use this form to specify options for the design to netlist and the views to netlist.


Netlist Setup

Netlisting Mode: Entire Design [✓ Incremental] Off

Netlist These Views: verilog pld_verilog lai_verilog lmsi_verilog schematic symbol

Netlist For LAI/LMSI Models ☐

Generate Verilog Test Fixture Template ✓

Netlist Uppercase	☐	Generate Pin Map	☐	Preserve Buses	✓
Netlist SwitchRC	☐	Skip Null Port	☐	Netlist Uselib	☐
Drop Port Range	✓	Incremental Config List	✓	Symbol Implicit	☐
Assign For Alias	☐	Skip Timing Information	☐	Declare Global Locally	☐
Netlist Explicitly	☐	Support Escape Names	☐	Single Netlist File	☐

Terminal SyncUp: Expand on Mismatch

Stop Netlisting at Views: verilog pld_verilog lai_verilog lmsi_verilog symbol

Global Power Nets: VDD!

Global Ground Nets: GND!

Global TimeScale Overwrite Schematic TimeScale ☐

Global Sim Time: 1 Unit: ns

Global Sim Precision: 1 Unit: ns

The following tables describe how the netlisting options affect netlisting.

Specifying the Netlisting Mode

Option	Description
Entire Design	Netlists your entire design regardless of which cells have been modified since the last netlist was generated.
Incremental	Netlist only those parts of the design that have changed since the last netlist was generated.
Off	Does not generate netlist of your design if another netlist exists.

Specifying the Views and Hierarchy to Netlist

Option	Description
Netlist These Views	Defines which view is netlisted for each cell or module. You use this option with the *Stop Netlisting at Views*. The netlister starts at the top-level cell in the design being simulated and works down the hierarchy, selecting the appropriate view for each cell netlisted. For each cell, the netlister searches for a view from the list, in left-to-right order, and netlists the first view it finds that is on the list. For example, if the list were *`behavioral verilog schematic symbol`*, the netlister first searches for a *`behavioral`* view, then a *`verilog`* view, and so on. If the current cell contains only a *`schematic`* and a *`symbol`* view, the netlister would netlist only the *`schematic`* view.

Generating Default Test Bench and Stimulus

Option	Description
Generate Verilog Test Fixture Template	Specifies whether or not you need the netlister to generate the test fixture template.

Controlling Netlisting

Option	Description
Netlist For LAI/LMSI Models	Netlister uses all `Lai_verilog` or `lmsi_verilog` cellviews for this simulation. The Lai_verilog property values indicate the use of SmartModels Library. The lmsi_verilog property values indicate the use of LM-family hardware simulation models in your analysis. Cells that do not have the `Lai_verilog` or `lmsi_verilog` view type are netlisted according to the priorities established with the *Netlist These Views* and *Stop Netlisting at Views* options. **Note:** To attach the `lai_verilog` or `lmsi_verilog` view property to an instance on a VSE schematic, use the *Edit – Properties* command. The *Edit – Properties* command appears on the schematic window. Refer to the *Virtuoso Schematic Editor L User Guide* for more information about using the *Edit – Properties* command to attach an `lai_verilog` or `lmsi_verilog` view property.
Netlist Uppercase	Generates a netlist in all uppercase letters. This option creates compatible modules that are disparate in case.
Generate Pin Map	Instructs the netlister to create the pin mapping files necessary to convert Standard Delay Files (SDF) pin names to NC-Verilog pin names. Select this option only when you are ready to backannotate. This option is required when the pin names for a symbol in your schematic differ from those in the Verilog library model description. After you create your pin map, your entire design is netlisted automatically to ensure that the netlister creates pin maps for the entire design. *Caution* ***The Generate Pin Map option must remain selected on subsequent runs. Otherwise, the netlister deletes your pin map directory.***

Controlling Netlisting

Option	Description
Preserve Buses	Instructs the netlister to preserve buses (vectors) in the resulting netlist. If this option is off, the netlister expands vector nets to single-bit equivalents (scalars) in the resulting netlist. For more information on bus constructs, refer to Bus Constructs on page 41.
Netlist Switch RC	Specifies that netlisting includes user-defined RC switch properties.
Skip Null Port	Specifies that netlisting ignores floating instance ports.
Netlist Uselib	Specifies that the netlister automatically adds the `'uselib` directive to the netlist when a design includes two similarly named cells from two different libraries. For detailed information, refer to Adding Directives on page 42.
Drop Port Range	Prints the module port without the port range.
Incremental Config List	Writes the renetlisted cellviews to the Configuration List.
Symbol Implicit	Suppresses printing of the net name during instance port formatting.
Assign For Alias	When enabled, specifies that the netlister uses an assignment statement for patches between nets. When disabled, specifies that the netlister applies the default `cds.alias` to patches between nets.
Skip Timing Information	When enabled, causes the netlister to ignore timing information assigned to instances in the design.
Netlist Explicitly	Generates name-based port lists (using connection-by-name syntax) for modules in Verilog views. Turn this option off to generate order-based port lists.This option does not apply to instances whose master module is generated by the netlister. For example, the netlister converts all schematics into Verilog modules. For instances of these modules, the netlister always creates order-based port lists.
Support Escape Names	When enabled, causes the netlister to include escaped names in the netlist.

Controlling Netlisting

Option	**Description**
Single Netlist File	When selected, generates a single verilog netlist instead of multiple netlists (one for each module). The netlist file is generated in the current simulation run directory with the name `netlist`.
Stop Netlisting at Views	Controls the level of hierarchy at which netlisting stops. After netlisting a cell, the netlister checks whether the view netlisted is on this stop list. If it is, the netlister stops expansion of the design for this cell. The order of views in the stop list is irrelevant. For example, if the stop list is *`functional verilog symbol`*, the netlister checks each netlisted cell to determine if it contains a *`functional, verilog,`* or *`symbol`* view. If it does, the netlister stops netlisting for that cell. If not, it goes on to the next cell in the design library.
Terminal SyncUp	Specifies how to synchronize terminals between an instance and its switched master. You can choose from the following three options: ■ *Expand on Mismatch* (default) retains the design terminals as is, unless there is a mismatch. In case of a mismatch, the netlister expands the mismatched terminals. Use this option to generate a pure explicit netlist with flattened buses. ■ *Honor Switch Master* always honors the switch master terminals. ■ *Merge All* merges all terminals to create simple scalar and pure bus terminals. The resulting netlist does not have any bundles or split buses. **Note:** You can also set the `hnlVerilogTermSyncUp` variable in the `.vlogifrc` file. The possible values are `mergeAll`, `honorSM`, and `nil` (default). For more details, refer to Synchronizing Terminals on page 114.

Defining Global Power and Ground Signals

Option	**Description**
Global Power Nets	Specifies the global net names you want netlisted with the supply1 wire type. Supply1 wire types are driven to logic state 1. The net names you specify must conform to global naming conventions as described in the *Virtuoso Schematic Editor L.*
Global Ground Nets	Specifies the global net names you want netlisted with the supply0 wire type. Supply0 wire types are driven to logic state0. The net names you specify must conform to global net naming conventions as described in the *Virtuoso Schematic Editor L.*
Global TimeScale Overwrite Schematic TimeScale	Specifies that the defined Global Time Scale variables (described below) overwrite any time values or units defined within a schematic. The results may vary depending on the various factors. For more information, refer to Results Dependency on page 42.
Declare Global Locally	When enabled, lets you declare global signals locally. When disabled, causes the netlister to use the default signals (Global Power Nets and Global Ground Nets).
Global Sim Time	Specifies a value for the global simulating time
Unit	Specifies the units for the global simulation time value. The valid values are *`s, ms, us, ns`*, and *`ps`*.
Global Sim Precision	Specifies a global precision value for the global simulation time

Bus Constructs

Bus constructs that cannot be represented with Verilog bus constructs are represented by using in-line concatenation, a Verilog feature. In some cases, bus names that are bundled or that are part of aliasing schemes are mapped to new names because NC-Verilog does not support certain bundling schemes. Mapped bus names are prefixed by `cdsbus` and have a sequence number assigned. The sequence number is a function of how many names are mapped. The following examples illustrate how bundled signals are mapped:

- bus `<a,b[1]>` maps to `cdsbus <[0:1]>`; `a` as bit 0 and `b [1]` as bit 1
- bus `<0:8:2>` maps to `cdsbus1<0:4>`

Adding Directives

The netlister automatically adds the `'uselib` directive only when the stop view for a cell is *`functional`* or *`behavioral`* or *`system`*. When the stop view for a cell is `verilog` or `symbol`, you must add the `lmoToolPath` property to the library that contains the cell. Refer to the following steps for adding directives:

- Choose the *Tools – Library Manager* command from the CIW menu banner.
- Specify the library by clicking on the library name in the Library Manager window.
- Choose the *Edit – Properties* command from the Library Manager menu banner.
- Select the *Add* command from the Library Property Editor.
- In the Add property form, enter the values as shown below:
    - ❑ Property Name: *`lmoToolPath`*
    - ❑ Property type: *`ILList`*
    - ❑ Property value: *`(verilog((“../vsyn/verilog”)))`*
- Click *OK* on the Add Property Form.
- Click *OK* on the Library Editor form.

The *`lmoToolPath`* property is added to the specified library.

> **Note:** You cannot netlist two similarly named cells from two different libraries when a cell has a `schematic` stop view.

For more information about adding library properties, refer to the *Library Manager User Guide*.

Results Dependency

The results for *Global TimeScale Overwrite Schematic TimeScale* option vary depending on the following:

- Whether this option is enabled
- Whether your schematic has an assigned time scale (that is, any time scale values or units

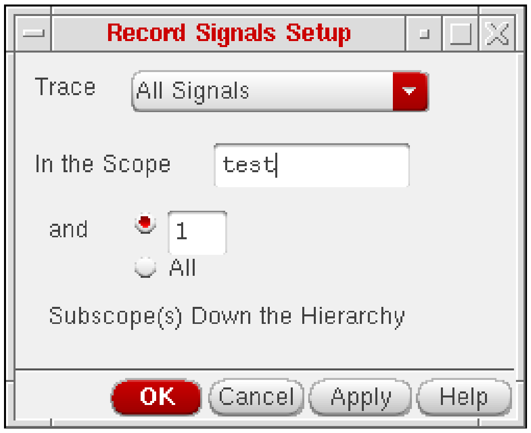
The table below describes the possible outcomes in various conditions:)

	Option Enabled	**Option Disabled**
Schematic has an assigned time scale	■ The system overwrites all assigned time scale values with the defined global time values. ■ The system converts all assigned time units to the defined global time unit scale.	■ The system does not overwrite any time scale settings within the schematic.
Schematic does not have an assigned time scale	■ The system assigns the defined global time scale values. ■ The system converts all assigned `ns`, `ps`, and `fs` time units to the defined global time unit scale. ■ The system appends the default time unit (1ns) to any values without time units.	■ The system does not overwrite any assigned time scale values. ■ The system converts any assigned `ns`, `ps`, and `fs` time units to the defined global time unit scale. ■ The system appends the default time unit (1ns) to any values without time units.

For more information on netlisting, refer to Chapter 5, “Netlisting.”

Record Signals

Specifies a set of signals to trace and save in the SHM database. The SHM database stores waveform displays. On selecting this option, the Record Signals Setup form is displayed.



Trace: All Signals
In the Scope: test
and [✓ 1] All
Subscope(s) Down the Hierarchy

The following table gives the description of the various options on the *Record Signals Setup* form:

Record Signals Setup Options

Option	Description
Trace	Tells the system to probe a set of signals. You specify the set of signals with the cyclic button. For information on signal sets, refer to Signal Sets on page 45
In the Scope	Specifies the scope of the verilog design under simulation in which the specified object(s) is to be probed and how many scope levels to descend when searching for objects to probe if a scope is specified. You must specify one of the following arguments: ◆ *n*, to descend the specified number of scopes. For example, 1 means include only the given scope, 2 means include the given scope and its subscopes, and so on. The default is 1. ◇ *All*, to include all scopes in the hierarchy below the specified scope(s)

Signal Sets

The following table describes the various signal sets you can specify in the *Trace* option:

Signal Sets Options

Option	Description
All Signals	Traces all signals in your design. This specifies that all of the declared objects within a scope, except for VHDL variables, are to be included in the probe.
All Input Ports	Specifies that all inputs within a scope are to be included in the probe.
All Output Ports	Specifies that all outputs within a scope are to be included in the probe.
All Ports	Specifies that all ports within a scope are to be included in the probe.
Only Specified Signals	Specifies that only the scopes specified are to be included in the probe.

Note: The object(s) being traced in the *All Signals, All Input Ports, All Output Ports, All Ports* options apply to:

- The scope(s) named in an argument
- The subscopes specified with the *In the scope* option
- The current debug scope (if no scope(s) or object(s) is named in an argument)

Record Signals for Batch Simulation

Signals are saved only when the simulation is run in the interactive mode. In order to save signals in Batch Mode, you should perform the following steps:

- Create a new file and fill in the kind of signals to probe following:
    - For Top Level I/O

      ```
      database -open shmWave -shm -default -into shm.db
      probe -create -shm test -all -depth 1
      run
      ```

    - For All Signals

```
database -open shmWave -shm -default -into shm.db
probe -create -shm -all
run
```

- ❑ For Selected Signals

```
database -open shmWave -shm -default -into shm.db
probe -shm <signal_names>
run
```

where, `<signal_names>` is the space separated list of names of signals to probe.

- Create another file with the following line:

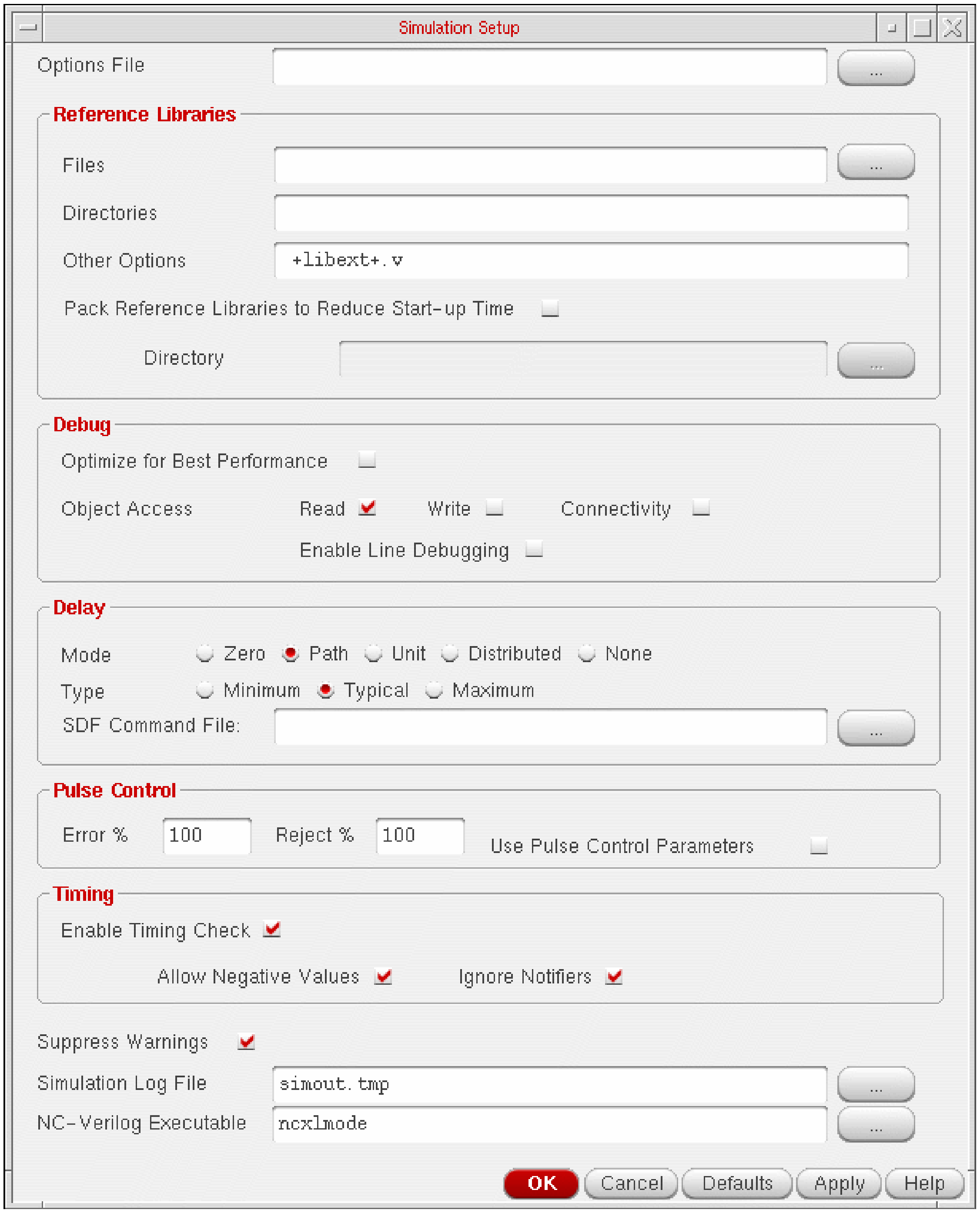
```
+ncsimargs+" -input file_name"
```

replace `file_name` with the full path name of the file created in the first step.

- In the NC Vlog Integ *Simulation Setup* form, enter the name of the file created in the second step, in the *Options File* field.
- These steps will enable you to create an `SST2` database of the name `shm.db` in the run directory. You can then view the waveforms by invoking either SimVision or Signalscan from shell.

Simulation

Selecting this option displays the Simulation Setup form. This form lets you specify various options that control and affect the simulation.



Simulation Setup

Options File:

Reference Libraries

Files:
Directories:
Other Options: +libext+.v
Pack Reference Libraries to Reduce Start-up Time ☐
Directory:

Debug

Optimize for Best Performance ☐
Object Access: Read ✓ Write ☐ Connectivity ☐
Enable Line Debugging ☐

Delay

Mode: Zero [✓ Path] Unit Distributed None
Type: Minimum [✓ Typical] Maximum
SDF Command File:

Pulse Control

Error % 100 Reject % 100 Use Pulse Control Parameters ☐

Timing

Enable Timing Check ✓
Allow Negative Values ✓ Ignore Notifiers ✓

Suppress Warnings ✓
Simulation Log File: simout.tmp
NC-Verilog Executable: ncxlmode

The following tables describe how to complete the Simulation Setup form by specifying the options listed below:

Simulation Options

Option	**Description**
Options File	Specifies an ASCII file containing NC-Verilog invocation command options. The file can also contain source text filenames or NC-Verilog predefined **+** or **-** options that are not available through the interface.
Suppress Warnings	Suppresses all warning messages during your NC-Verilog simulation.
Simulation Log File	Specifies the name of the file in the current run directory in which you want to store the simulation log. The default file is simout.tmp.

Reference Libraries

Option	**Description**
Files	Identifies vendor-supplied Verilog ASIC libraries. NC-Verilog also scans this file for module and UDP definitions that have not been resolved in the normal source text files specified on the command line. This option takes any valid operating system file for an argument.
Directories	Specifies the path (or paths) to library files that contain module and UDP definitions. NC-Verilog searches these directories when it compiles your netlist and functional views.
Other Options	Specifies the extensions, such as `.v`, of files that you want to refer to in the specified directories while simulating a design.
Pack Reference Libraries to Reduce Start-up Time	This option is enabled once you specify a reference library name. You can use this option to specify a file path in which the reference library files/directories will be compiled. A packed reference library reduces the simulator start-up time significantly and it is useful for those reference libraries that are large in size and are changed rarely. On recompilation, a packed library needs to be unpacked, compiled, elaborated and repacked. This is required only if the library files/directories have been modified. You can confirm the unpacking and repacking of reference library by using the Repacking Library dialog box.

Debug Options

Option	**Description**
Optimize for Best Performance	Sets the visibility access for all objects in the design. This is used to increase the simulation performance by not giving read, write or connectivity access to the simulation objects. By default this button is disabled.
Object Access	Specifies the read, write and connectivity access of the simulation object. The default value is read only access. For information, refer to **Object Access Options** on page 51

Debug Options

Option	**Description**
Enable Line Debugging	It set to use line breakpoints and single stepping. If this option is not set, line breakpoints and single stepping would still not be available even when you select read, write and connectivity access.

Delay Options

Option	**Description**
Mode	Globally alters the delay values specified in your design. You can filter results by setting the following options: ◇ Zero ignores all module path delays, timing checks, and structural and continuous assignment delays. ◆ Path uses delay information from specified blocks that contain module path delays. It ignores structural and continuous assignment delays with the exception of `trireg` charge decay times. ◇ Unit ignores module path delays and timing checks and converts all structural and continuous assignment delays that are nonzero to one time unit. ◇ Distributed uses the delay on nets, primitives, and continuous assignments. It ignores module path delays. ◇ None uses all of the delays in your netlist.
Type	Specifies the delay type. You can filter results by setting the following options: ◇ Minimum selects all minimum delays. ◆ Typical selects all typical delays. ◇ Maximum selects all maximum delays.
SDF Command File	Specifies the name of the SDF command file that you need ncelab to annotate. If an SDF delay file already exists, then the system automatically generates an SDF command file and populates the filename into this field.

Object Access Options

The following table describes the access rights for the *Object Access* option:

Object Access Options

Option	Description
Read	It is used if you need to probe objects in the design and generate an SHM or VCD database. This lets you use SimVision or Signalscan to view waveforms or SimCompare to compare databases.
Write	It is used if you need to specify values using the interactive simulation interface. For example when you use the *force* or *deposit* commands from the interactive simulation interface.
Connectivity	It is used to if you need to display the load or driver information. For example, this information is required by the driver command and by the Signal Flow Browser.

Pulse Control Options

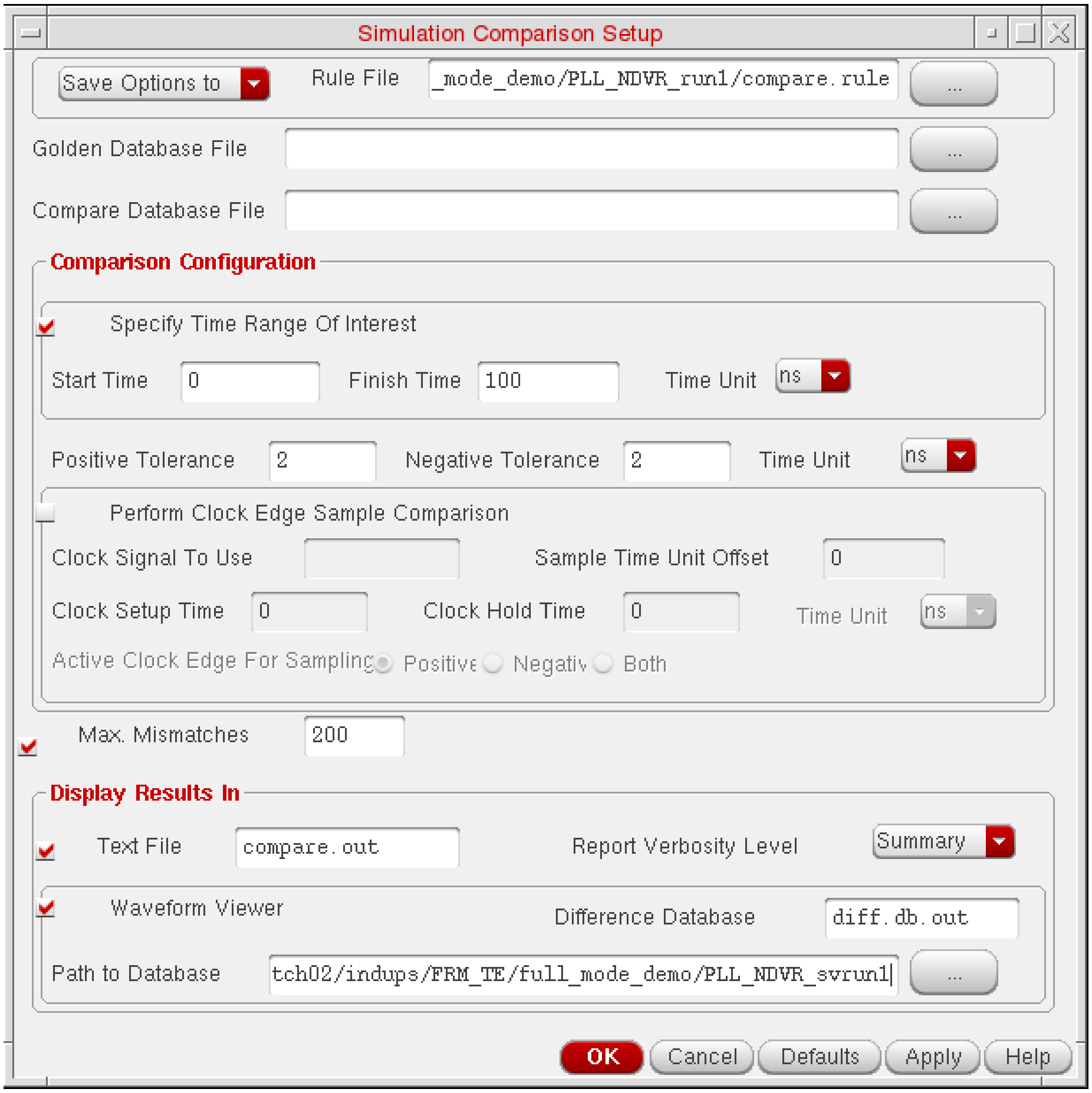
Option	Description
Error%	Sets the error limit to one of the three values: 0, 50, or 100 percent. The default setting for NC-Verilog is 100. **Note:** A pulse that is less than or equal to this specified percentage, but greater than the reject limit, sets the module path output pulse to the *e* logic value.
Reject%	Sets the reject limit to one of three values: 0, 50, or 100 percent. The default setting for NC-Verilog is 100.
Use Pulse Control Parameters	Enables you to use `PATHPULSE$` special parameters to override the global pulse control.

Timing Options

Option	**Description**
Enable Timing Check	It is set to use the two timing check options which are disabled by default.
Allow Negative Values	It is set to allow negative values in `$setuphold` and `$recrem` timing checks in the Verilog description and in `SETUPHOLD` and `RECREM` timing checks in the SDF annotation. If this option is not set, any negative values in the Verilog description or in the SDF annotation are set to 0 and a warning is generated.
Ignore Notifiers	It is set to ignore notifiers in timing checks.

Simulation Compare

Selecting this option displays the Simulation Comparison Setup form.



Rule File: _mode_demo/PLL_NDVR_run1/compare.rule
Golden Database File:
Compare Database File:
[✓] Specify Time Range Of Interest
Start Time: 0
Finish Time: 100
Time Unit: ns
Positive Tolerance: 2
Negative Tolerance: 2
Time Unit: ns
[ ] Perform Clock Edge Sample Comparison
Clock Signal To Use:
Sample Time Unit Offset: 0
Clock Setup Time: 0
Clock Hold Time: 0
Time Unit: ns
Active Clock Edge For Sampling: [● Positive] ○ Negativ ○ Both
[✓] Max. Mismatches: 200
[✓] Text File: compare.out
Report Verbosity Level: Summary
[✓] Waveform Viewer
Difference Database: diff.db.out
Path to Database: tch02/indups/FRM_TE/full_mode_demo/PLL_NDVR_svrun1

The following tables describe the various form options:

File Options

Option	**Description**
Rule File	Specifies the name of comparison rule file. You can also specify an option for the system to create a comparison rule file by reading the entries in the form. Alternatively, you can supply your own rule file in which case, the system reads into your rule file to drive the simulation comparison.
Golden Database File	Specifies the golden file to be used for comparison
Compare Database File	Describes the contents of the current director. The current directory contains the comparison SHM database results, which you compare to the golden database results described in the previous option.

Comparison Configuration Options

Option	**Description**
Specify Time Range of Interest	On selection, enables you to specify the time range that contains the signal events that you want to compare. Use absolute time values to define the range. For more information on its options, refer to Timing Options on page 52
Perform Clock Edge Sample Comparison	On selection, enables you to perform a clock edge sample comparison. For more information on its options, refer to Clock Edge Sampling Options on page 55
Max. Mismatches	When enabled (default), requires you to enter a number in the associated text field that specifies the number of mismatches that you want the system to find. The system stops the comparison when the specified limit is reached. When disabled, the system ignores the associated text field and continues the comparison regardless of the number of mismatches found.

Timing Options

Option	**Description**
Start Time	Specifies when the simulation comparison begins.

Timing Options

Option	Description
Finish Time	Specifies when the simulation comparison ends.
Time Unit	Displays the units of measure for the time range. You cannot modify this value.
Positive Tolerance	Specifies the positive tolerance for a matching event. This is the time window after a golden event during which a test event may match.
Negative Tolerance	Specifies the negative tolerance for a matching event. This is the time window after a golden event during which a test event may match.
Time Unit	Displays the units of measure for the *`Cycle Time`*, *`Strobe Offset`*, and *`Strobe Width`* fields. You cannot modify this value.

Clock Edge Sampling Options

Option	Description
Clock Signal to Use	Specifies the clock signal to be used in the design
Sample Time Unit Offset	Specifies the number of time units by which the sampling times for the comparison and the relative times at which setup and hold timing checks occur are offset from the edge of the clock signal. The default value is 0. This means that each sample is taken at the clock edge and all setup and hold checks are performed relative to the clock edge.
Clock Setup Time	Specifies the setup time for the clock signal used.
Clock Hold Time	Specifies the hold time for the clock signal used.
Time Unit	Displays the units of measure for the *`Cycle Time`*, *`Strobe Offset`*, and *`Strobe Width`* fields. You cannot modify this value.
Active Clock Edge for Sampling	Specifies the active edge of the clock signal at which sampling occurs. It can be positive, negative or both.

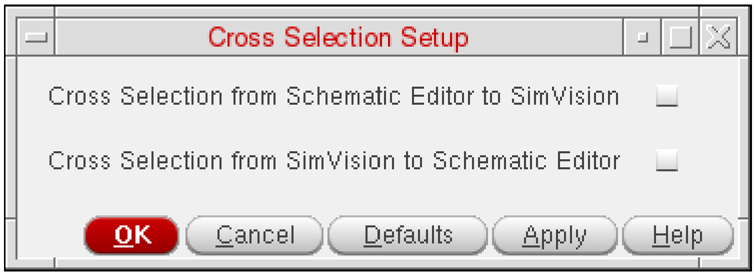
Display Results In Options

These options let you manage the display of comparison results. When there are mismatches, you can specify that the software display the results in either an STV window or a SignalScan window, or both. When there are no mismatches, neither window appears. The following table describes the various options:

Option	**Description**
Text File	Specifies that you want the software to store the comparison results in the text file displayed in the text field. **Note:** From Incisive Unified Simulator 82 (IUS82), the default name of the text file is `compare.out`. For previous IUS versions, the default name is `compareScan.out`. When you enable this option (default), the software automatically opens the file in an STV (Source Text Viewing) window after completing the comparison. However, the software does not open the file if there is a mismatch in the simulation results.
Report Verbosity Level	Specifies the level of detail for the text file. You can choose the following from the drop down list: ■ Summary - summarizes the mismatch information ■ Detailed - writes a detailed report for each mismatch
Waveform Viewer	Specifies that you want the software to store the comparison results in a waveform database. When enabled (default), the software automatically opens a SimVision or Signalscan window that displays the waveforms after completing the comparison. However, the software does not display the waveforms if there is a mismatch in the simulation results.
Difference Database	Specifies the waveform database where the software stores the comparison results.
Path to Database	Specifies the path to the directory where the difference database resides.

Cross Selection Setup

Selecting this option displays the Cross Selection Setup form.

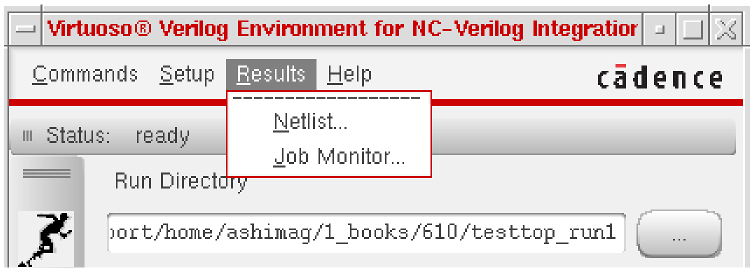

Cross Selection Setup
Cross Selection from Schematic Editor to SimVision
Cross Selection from SimVision to Schematic Editor
OK
Cancel
Defaults
Apply
Help

The following table describes the various form options:

Cross Selection Options

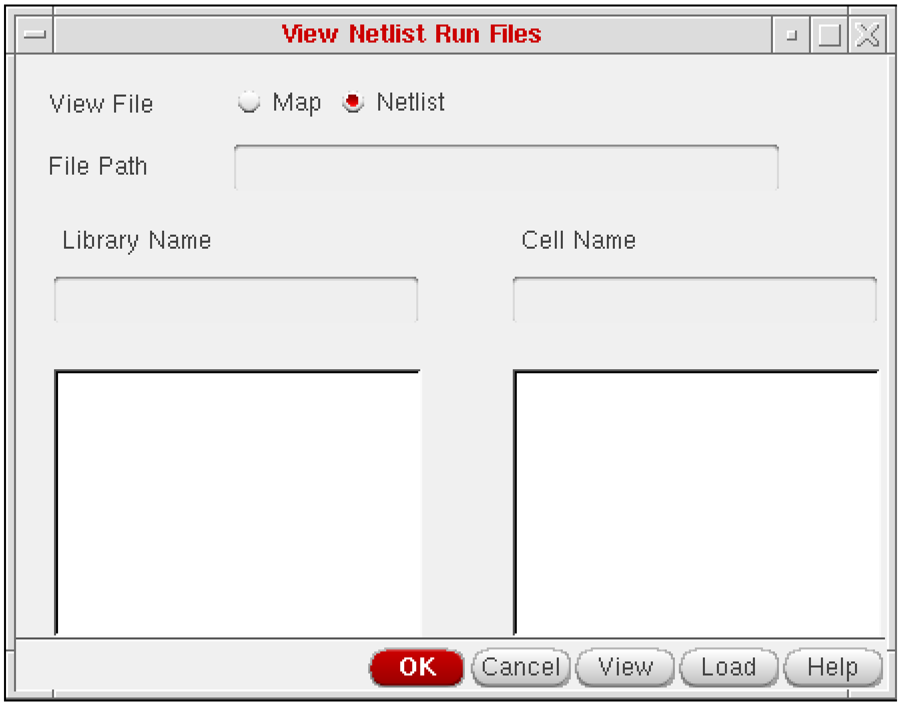
Option	Description
Cross Selection from Schematic Editor to SimVision	On selection, enables you to select an object in the schematic design in Virtuoso Schematic Editor and have the corresponding simulation object selected in SimControl.
Cross Selection from SimVision to Schematic Editor	On selection, enables you to select a simulation object in SimControl and have the corresponding object selected in the schematic design in Virtuoso Schematic Editor.

Results Menu


Virtuoso® Verilog Environment for NC-Verilog Integratior
Commands
Setup
Results
Help
cadence
Status: ready
Netlist...
Job Monitor...
Run Directory
port/home/ashimag/1_books/610/testtop_run1
...

Netlist

Displays the netlist or a map file in a view window. The netlist file contains the Verilog text description of your design. A map file describes how the system maps names between Cadence® Verilog and the Design Framework II environment. When you netlist your design, the netlister generates a netlist and a map file. This command opens the View Netlist Run Files form that you use to specify the type of file and the name of the file you want to view.



View File: Map [✓ Netlist]

Note: If the View Netlist Run Files form is blank or does not display all the libraries you want to see, click *`Load`*.

The following table describes the various form options:

View Netlist Run Options

Option	Description
View File	Specifies which type of file that you want to view. You can select one of the following: ◇ Map specifies that you want to view a map file. ◈ Netlist (default) specifies that you want to view a netlist file. **Note:** The netlist or a map files are displayed in a view window. For more information about a view window, see “View Windows”
File Path	Displays the full path to the library and cellview you select from the list boxes. You cannot edit this field.
Library Name	Displays the design library you select from the list box below this field. You cannot edit this field.
Cell Name	Displays the cell you select from the list box below this field. You cannot edit this field.

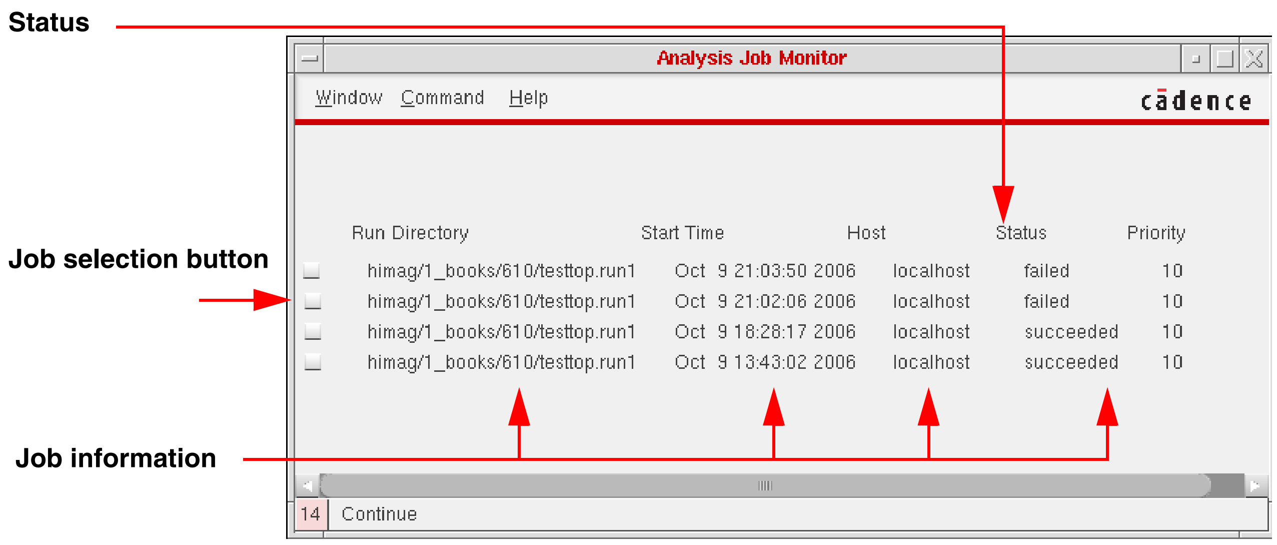
Job Monitor

Displays the Analysis Job Monitor window, which you can use to select options that control batch jobs or to view information about batch jobs.

Analysis Job Monitor Window

On selecting *Job Monitor* option, the Analysis Job Monitor window appears. The window displays the following:

- The status of any batch job you started during this session
- Option buttons that let you control the simulation
- Selection buttons that you use to specify individual batch jobs
- Information about a batch job and the progress of a batch simulation



	Run Directory	Start Time	Host	Status	Priority
☐	himag/1_books/610/testtop.run1	Oct 9 21:03:50 2006	localhost	failed	10
☐	himag/1_books/610/testtop.run1	Oct 9 21:02:06 2006	localhost	failed	10
☐	himag/1_books/610/testtop.run1	Oct 9 18:28:17 2006	localhost	succeeded	10
☐	himag/1_books/610/testtop.run1	Oct 9 13:43:02 2006	localhost	succeeded	10

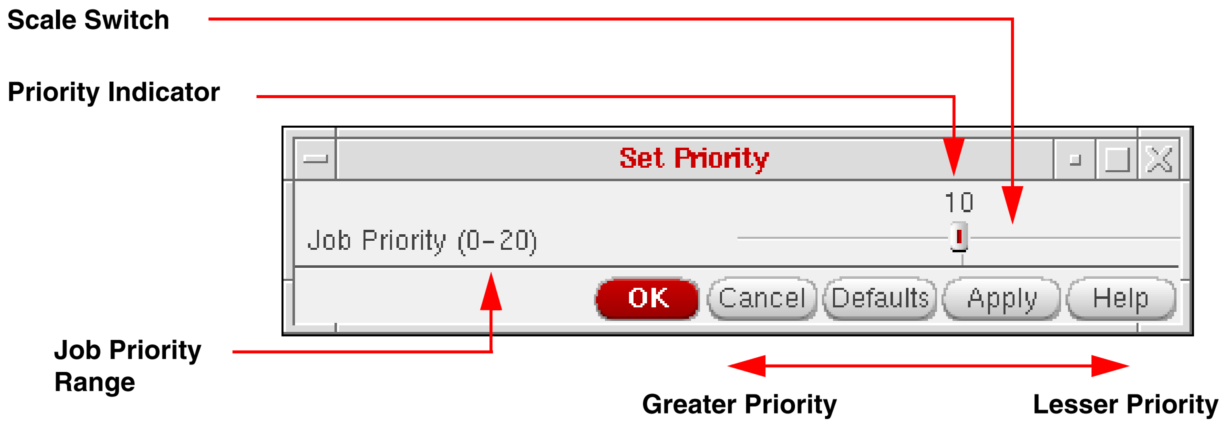
The Analysis Job Monitor form lists all batch simulations started during the current session.

- Run Directory specifies the run directory where the batch simulation is running.
- Start Time specifies the date and time the batch simulation was started.
- Host specifies the name of the host where the simulation is running.
- Status specifies the status of the batch simulation. The status of a batch job can be `running`, `suspended`, `succeeded`, `killed`, or `failed`.
- Priority specifies the order in which a batch job is processed.

Using Job Monitor

The option buttons at the top of the window let you control batch simulations. To apply an option to a batch job, click the job selection button for a batch job and then click an option button.

- Show Run Log opens a view window that displays the simulation interface log file, `si.log`, for the selected job.
- Set Priority displays the *Set Priority* form. You can use the S*et Priority* form to specify the priority of the selected job by dragging the scale switch left or right. The number above the scale switch changes as you drag the scale. A larger number indicates a lesser priority. When you decrease the priority of a particular batch job, the system can process jobs with higher priorities faster.



- Kill terminates the selected job, whether it is running or suspended, and sets the batch status to `kill`. A dialog box requests confirmation that you want to terminate the job.
- Suspend stops the selected job and sets the batch status to `suspended`. You can resume a suspended job by clicking the *Continue* button.
- Continue resumes a job you suspended by clicking `Suspend`.
- Remove Entry deletes the selected job from the Analysis Job Monitor window but does not terminate or suspend it. A dialog box prompts you for confirmation that you want to delete the job from the window.

Help Menu

The *Help* menu lets you access help on using NC-Verilog Integration Environment. You can also access online support and resources from this menu.

Fixed Menu

This section describes each fixed menu command. Until you start an interactive simulation, Start Interactive is the only fixed menu command you can access.

Initialize Design	starts the NC-Verilog simulator. You must initialize design before you can access the other fixed menu commands.
Generate Netlist	triggers the OSS netlist() routine to generate an incremental hierarchical netlist in the specified run directory.
Simulate	simulates the design in either interactive or batch mode.
Simulation Compare	invokes the SimCompare tool to compare the golden and test SST2 databases.

Customizing Command Form Options

The default option settings for the NC-Verilog forms are set by the system when you enter an empty run directory. The initial settings are supplied with the software. When you edit the default settings on a form, the system saves the new settings to the `.vlogifrc` file. The `.vlogifrc` file settings override the default setup options supplied with the NC-Verilog software or supplied by a user in an .simrc file.

The `.vlogifrc` file is rewritten whenever you complete one of the following actions:

- Exit the run directory
- Choose the File - Quit command from the AEncv window
- Change to a new run directory

Forms that you can customize

In general, you can customize some of the options for each of the following forms:

- SDF Delay Annotation Setup

- Netlist Setup
- Record Signals Setup
- Simulation Setup
- Simulation Comparison Setup
- Cross Selection Setup

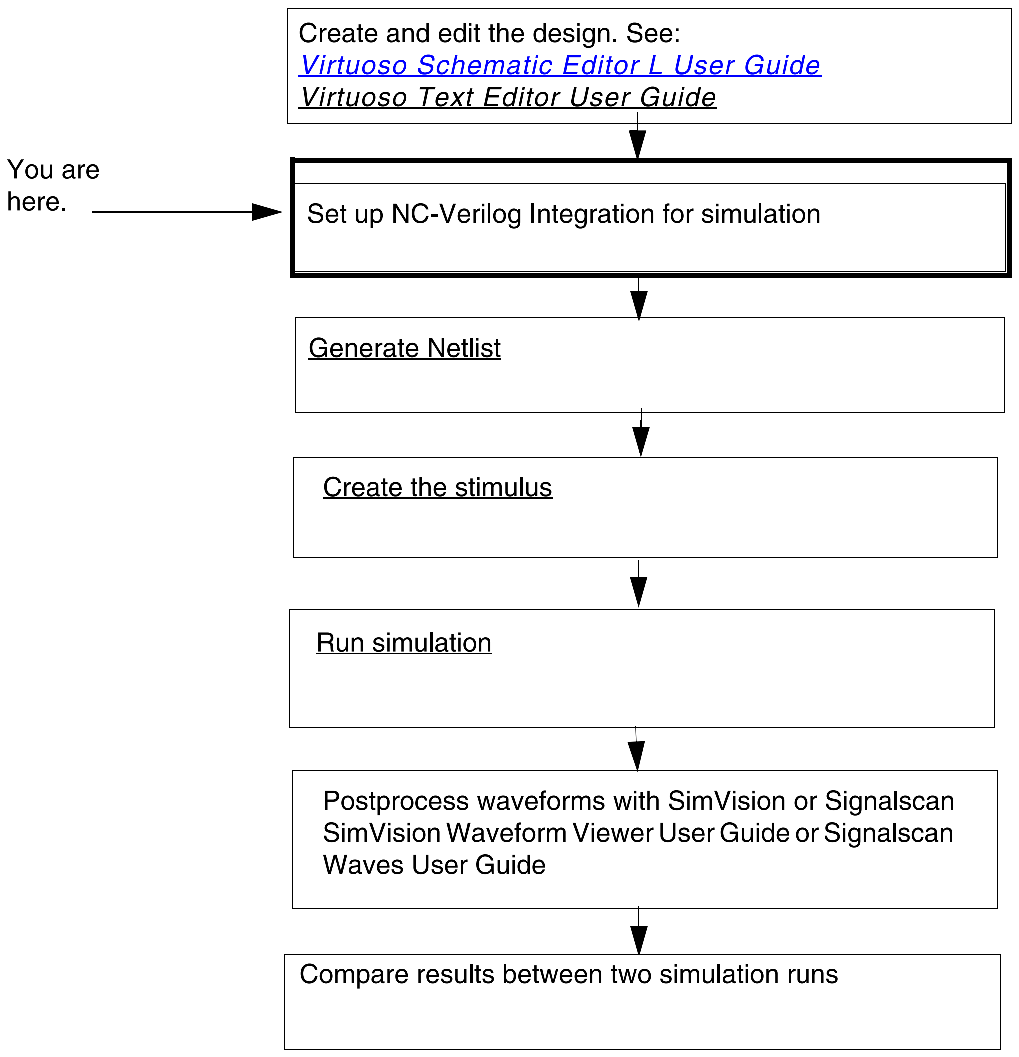
For more information, refer to Appendix A, “Customizing Your Environment”.

3

Setting Up the Simulation Environment

This chapter describes how to set up the NC-Verilog Integration Environment for a simulation.

Simulation Process Flowchart


Create and edit the design. See:
Virtuoso Schematic Editor L User Guide
Virtuoso Text Editor User Guide
You are here.
Set up NC-Verilog Integration for simulation
Generate Netlist
Create the stimulus
Run simulation
Postprocess waveforms with SimVision or Signalscan SimVision Waveform Viewer User Guide or Signalscan Waves User Guide
Compare results between two simulation runs

You can develop designs using Virtuoso Schematic Editor for schematics, and Virtuoso Text Editor for text cellviews. You can also use other Virtuoso tools, such as VHDL Import, to include cellviews for your designs.

Note: You can import Verilog, SystemVerilog, Verilog-AMS, VHDL, and VHDL-AMS text files into the DFII environment using the `cdsTextTo5x` command. This command also lets you generate the symbol views of the imported cellviews. For details, see *Importing Design Data by Using cdsTextTo5x*. You can also use this command to create text cellviews (5x structure), symbol views, and shadow database for SPICE, Spectre, DSPF, and PSpice.

Selecting a Design

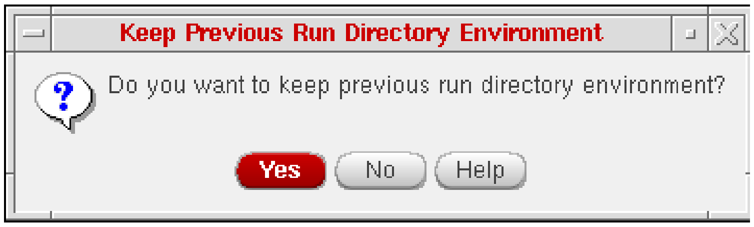
The process of selecting a design includes setting up your run directory and specifying the top level library, cell and the view names. You can specify this information in the main NC-Verilog window.

For more information on entering run directory name and top level design, refer to Run Directory on page 26.

Initializing a Design

To initialize a design, you can either use the Initialize design from Fixed Menu or select the Initialize design command from the Commands Menu in the main NC-Verilog window. Selecting this command initializes the specified run directory for netlisting and simulation after performing a set of checks on the run directory and the design. Following are the possible outcomes:

- If the specified run directory already exists, then the system loads `.vlogifrc` that exists in that directory.
- If the specified run directory does not exist, then the system loads the `$HOME/.vlogifrc`. In case no `.vlogifrc` exists, then the default settings are used.
- If you have netlisted and/or simulated in one run directory and then selected a new run directory, a dialog box prompts you to confirm if you want to use the previous run directory settings.



Keep Previous Run Directory Environment

Do you want to keep previous run directory environment?

- If you want to re-use an existing run directory and change the top level library or cell or view name, a dialog box is displayed that informs you that the run directory specified had been used for a different design and whether you would want to use the same run directory for the specified design.

Note: On initializing design, if the run directory already contains a netlist, then the *Generate Netlist, Simulate, Edit TestFixture* and *Simulation Compare* commands would be enabled. If the run directory does not contain a netlist, then the *Generate Netlist* and *Simulation Compare* commands would be enabled.

Netlisting a Design

Netlisting produces a Verilog text description of your design. The Verilog text description (the netlist) serves as the input for the NC-Verilog simulator. This section describes how to

- Specify Netlisting Options
- Run Hierarchical Netlister

Specify Netlisting Options

The Netlist Setup form lets you specify netlisting options before the simulator generates the netlist. To display the *Netlist Setup* form, select the *Netlist* command from the Setup Menu on the main NC-Verilog window.

The table below shows which options you set for specific netlisting tasks.

Mapping Tasks to Options

To Do This...	Use These Options...
Specify the Netlisting Mode	Entire Design Incremental Off
Specify the View and Hierarchy to Netlist	Netlist These Views Stop Netlisting at Views
Control Netlisting	Netlist for LAI/LMSI Models Netlist Uppercase Netlist Switch RC Drop Port Range Assign For Alias Netlist Explicitly Generate Pin Map Skip Null port Incremental Config List Skip Timing Information Support Escape Names Preserve Buses Netlist Uselib Symbol Implicit Declare Global Locally
Generate default verilog test bench and default verilog stimulus	Generate Verilog Test Fixture Template
Specify the Global Power and Ground Signals	Global Power Nets Global Ground Nets
Set Global Time Scale	Global Sim Time Global Sim Precision

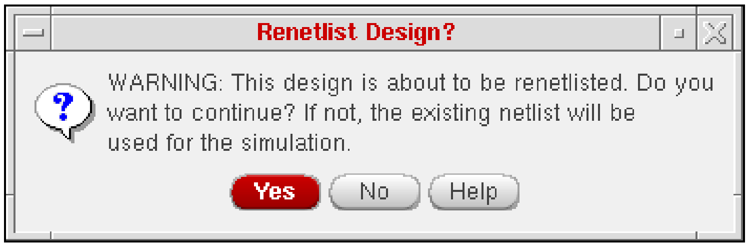
For more information on specifying the netlisting options, see Netlist on page 36.

Run Hierarchical Netlister

To generate an incremental hierarchical netlist in the specified run directory, you select either the *Generate Netlist* option from the Commands Menu or *Generate Netlist* option from the Fixed Menu.

The next action depends on how the *Netlisting Mode* option on the Netlist Setup form has been specified. (See Specify Netlisting Options on page 68 for more information about the Netlist Setup form.)

- If the *Netlisting Mode* option on the Netlist Setup form is disabled, and if the run directory already contains the netlist for the selected design, the Simulate command will be available in the Commands Menu and in the Fixed Menu so that you can proceed with simulating the design.
- If the *Netlisting Mode* option on the Netlist Setup form is enabled, and if no netlisting setup has been changed since the design was netlisted successfully, a dialog box appears, asking if you want to re-netlist the design or use the existing netlist for simulation.



WARNING: This design is about to be renetlisted. Do you want to continue? If not, the existing netlist will be used for the simulation.

If you click *Yes*, the simulator re-netlists the design. If you click *No,* the existing netlist is used to compile the design.

After successfully netlisting the design, the simulation specific menu commands that are still not available will be available.

Creating Stimulus Template File

By default, the system automatically creates the stimulus template file the first time you netlist your design in a new run directory. The default stimulus file, called `testfixture.verilog`, contains preinitialized signal settings that drive a simulation. This default operation is controlled by the *Generate Test Fixture Template* option on the Netlist Setup form.

You can edit the test fixture file using the Edit Test Fixture form. It can be accessed from the Commands Menu. This form gives you an option to either select the default stimulus file or test bench or specify another stimulus or test bench file location. You can also edit the existing test fixture using an editor. For more information on editing test fixture, refer to *Edit Test Fixture* option under Commands Menu on page 27.

Caution

If you select the Entire Design option in the Netlist setup form, and re-netlist a design, the system deletes the existing test bench and stimulus files and creates default test bench and stimulus files. If you need the edited files for later use, you need to save them to a different location before re-netlisting the entire design.

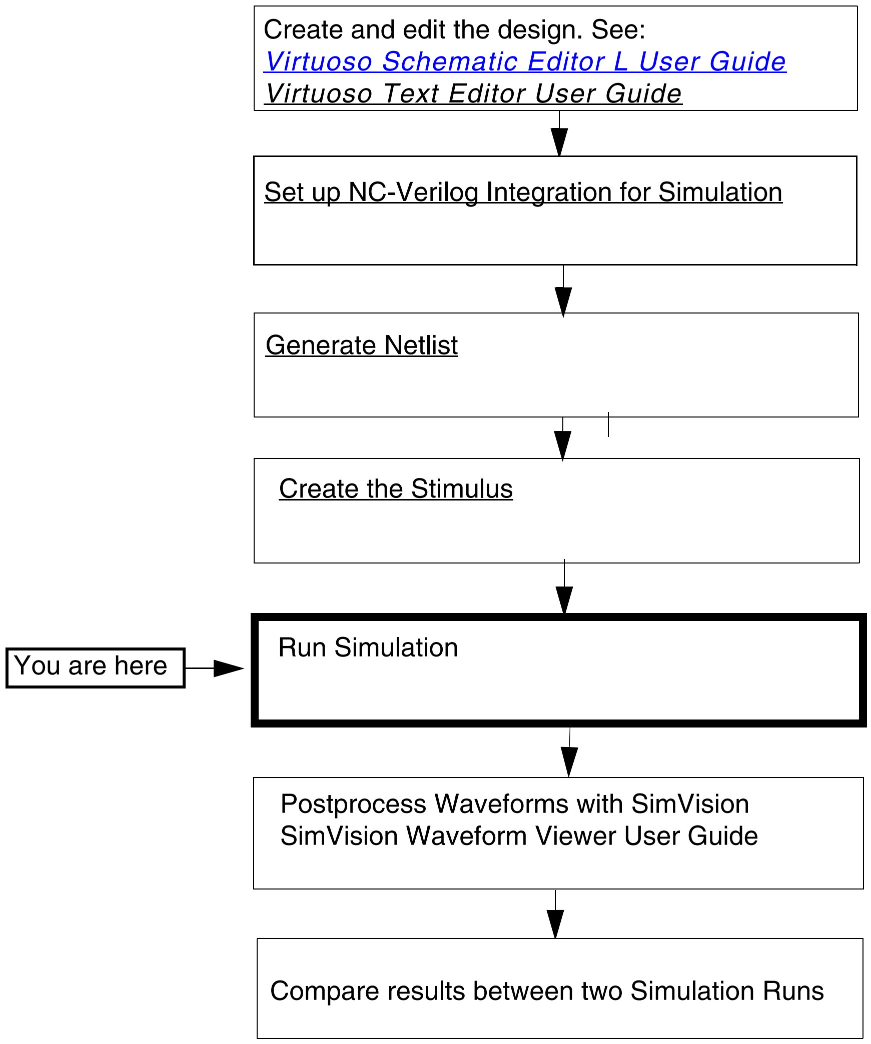
For more information on stimulus and test fixtures, refer to Chapter 6, “Working with the Stimulus” .

4

Running the Simulation

This chapter includes the following:

Simulation Process Flowchart


Create and edit the design. See:
Virtuoso Schematic Editor L User Guide
Virtuoso Text Editor User Guide
Set up NC-Verilog Integration for Simulation
Generate Netlist
Create the Stimulus
You are here
Run Simulation
Postprocess Waveforms with SimVision
SimVision Waveform Viewer User Guide
Compare results between two Simulation Runs

Introduction

In NC-Verilog Environment, you can run two types of simulation, namely:

- Interactive
- Batch

In Interactive simulation you can select the steps to Compile, Elaborate and, Simulate. After the design is successfully compiled and elaborated, the SimVision user interface opens. You can use this interface interactively to simulate and debug the design.

When you run a batch simulation, the Compile, Elaborate and, Simulate are run one after the other and do not require any user intervention.

Before you run a simulation, you must set the SDF delay annotation options and the simulation setup options.

Setting SDF Delay Annotation

In the NC-Verilog Integration Environment, you can annotate delays into your design either before you start simulation. This section describes how to read in a delay file at simulation time zero. The term time zero refers to the time before you start interactive or batch simulations.

Virtuoso Schematic Editor recognizes delay files written in SDF (Standard Delay Format). You can use SDF to create either a delay file for backannotation or a constraints file for forward annotation.

You set the SDF delay annotation using the SDF Delay Annotation Setup form. This form is accessed from the Commands menu. Once you specify the various options in the SDF Delay Annotation Setup form, the information you enter in the form is saved in the run directory and is applied each time you run a simulation.

Setting the Delay Annotation Options

This table shows you how to fill in the delay annotations on the Setup Environment form.

How Delay Annotation Options Affect Simulation

Option	Description
Suppress SDF CellType Checking	Turn this on to suppress the CellType mismatch checking performed by NC-Verilog SDF backannotator.
Suppress sdf2sdf3 Running	Turn this on to suppress the running of the sdf2sdf3 process when you start a simulation. When this option is on, the simulator reads the sdf2sdf3 information from the last simulation.
Suppress sdf2sdf3 Warning Message	Turn this on to suppress the warning messages generated by sdf2sdf3.
Delay File	Type in the path to your SDF delay file. If the file is in the run directory, you need only enter the file name.
sdf2sdf File	Type in the name of a file to store information generated by the sdf2sdf process. When you suppress the sdf2sdf process, the simulator reads information from the file specified in this field.
SDF Scope	Type in the name of the scope you want the system to preappend to the sdf2sdf file.
SDF Module Instance	Type in the name of the module that the SDF file annotates.
Delay Config File	Type in the name of your annotator configuration file, if you have one. Refer to the *SDF Annotator User Guide* for more information.
Log File	You can change the default log file name, which is `sdf.log`.
Use Delay Type	Select a delay type. This value overrides the type specified in the configuration file. If you select *Tool Control*, the software bases the delay type on the delay mode you enter on the Simulation Options form.

Option	**Description**
Scale source Delay	Select the type of delays from the delay file against which the scaling factor is applied. Your selection overrides the configuration file value (if any) unless you select *From Config*.
Scale Factor	Enter the scale factor to apply to the delays you selected in *Scale Source Delay*. Leave these fields set to 1 (the default value) if your delay file contains computed delay values. The SDF annotator uses these values to scale the minimum, typical, and maximum timing data from the SDF file before they are backannotated to NC-Verilog Integration. You must enter positive real numbers in the *Minimum*, *Typical,* and *Maximum* fields, for example, 0.5, 1.0, and 1.6.

Note: The *Use Delay Type*, *Scale Source Delay*, and *Scale Factor* options are used to determine the delays annotated to the Verilog design for simulation.

Setting Simulation Setup Options

You use the Simulation Options form to set options that control simulation. These options let you control acceleration, delays, and pulses. The form also lets you specify the Verilog libraries and the executable that you want to use.

You set the simulation options using the Simulation Setup form. This form is accessed from the Commands menu. Once you specify the various options in the form, NC-Verilog Integration sets up the simulation according to the options that you have specified.

The following tables describe how to complete the Simulation Options form by specifying the options.

Options File

Option	Description
Options File	Type in a path to an options file relative to the directory from which you invoked the Command Interpreter Window (CIW). The simulation run is based on the Verilog options in the file.

Reference Libraries Options

Option	Description
Library Files	Type in the paths to the library files containing third-party Verilog libraries used with this design.
Library Directories	Type in the paths to the dedicated library directories containing third-party Verilog libraries used with this design.
Pack Reference Libraries to reduce Start-up Time	Turn this option on to specify a file path in which the reference library files/directories will be compiled.

Debug Options

Option	Description
Optimize for Best Performance	Turn these options on to set the visibility access for all objects in the design. Object Access specifies the read, write and connectivity access of the simulation object. The default value is read only access.
Enable Line Debugging	Turn this option on to use line breakpoints and single stepping.7

Delay Options

Option	Description
Zero	Turn this option on if you do not want to apply delays.

Path	Turn this option on to use path (pin-to-pin) delays for objects that have both path and distributed delays attached to them.
Unit	Turn this on to apply one delay unit per gate.
Distributed	Turn this on to use distributed delays for objects that have both path and distributed delays attached to them.
None	Turn this on if you do not want to use all of the delays in your netlist.
Types	Specify a delay value to apply during simulation: minimum, typical, or maximum.
SDF Command File	Type in the paths to the SDF command file that you need ncelab to annotate.

Pulse Control Options

Option	**Description**
Error	Type in a percentage. The unknown value is assigned to the output pulse for any input pulse that is both shorter than or equal to this percentage of the module path delay, and greater than the reject percentage.
Reject	Type in a percentage. Input pulses that are shorter than this percentage of the module path delay are ignored.
Use PulseControl Parameters	Turn this option on to read in the PATHPULSE$ spec param from the Verilog model in the Verilog library, which overrides the error and pulse settings. (Verilog *+pathpulse* option)

Timing Options

Option	**Description**
Enable Timing Check	Turn this option on to specify the various types of timing checks.

Message Control Options

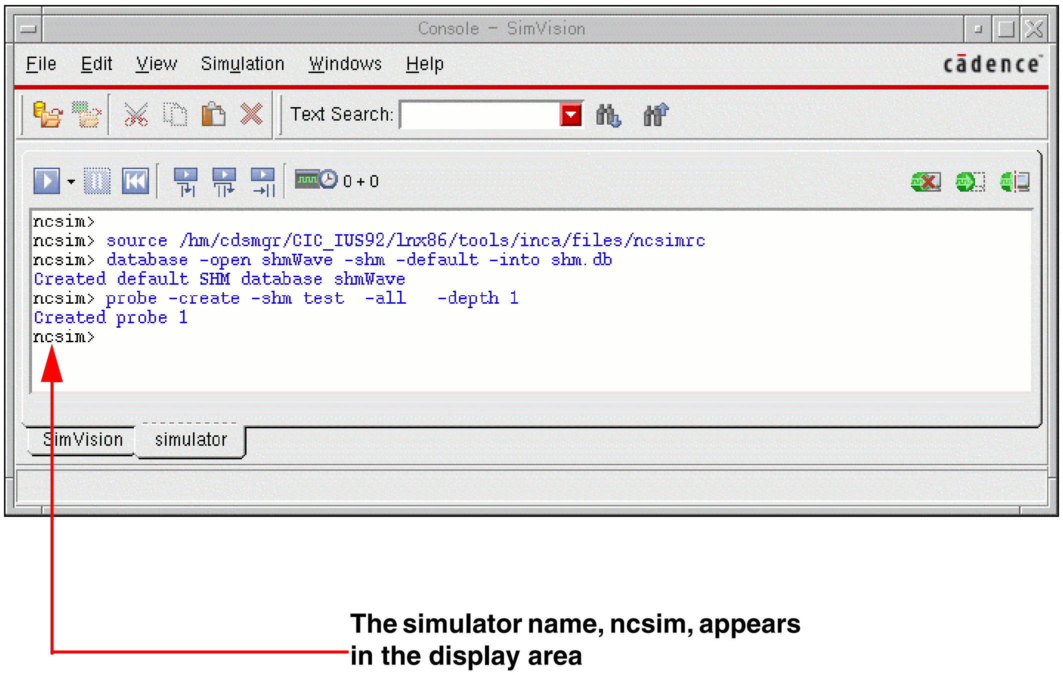
Option	**Description**

Suppress Warnings	Turn this option on to suppress warnings. For example, turn this button on if you do not want to be slowed down by warnings displays during a large simulation.

Simulating a Design

To simulate a design, you use the *Simulate* command in the Fixed menu. Alternatively you can also use the *Simulate* command from the *Commands* menu.

In interactive simulation, you select the option to compile, elaborate and simulate the design from the main NC-Verilog window. When you select all the three options, the system invokes SimVision's Design Browser and the Console windows after it successfully compiles and elaborates the design.



```
ncsim>
ncsim> source /hm/cdsmgr/CIC_IUS92/lnx86/tools/inca/files/ncsimrc
ncsim> database -open shmWave -shm -default -into shm.db
Created default SHM database shmWave
ncsim> probe -create -shm test  -all   -depth 1
Created probe 1
ncsim>
```

The simulator name, ncsim, appears in the display area

In the Design Browser window, you can interactively simulate and debug the design. Using the Design Browser window, you can directly interact with the simulator to open a database, trace signals, single step, set breakpoints, observe signals, and perform many other functions to verify your design. The Design Browser is part of the Cadence SimVision Analysis environment which is a unified graphical debug environment for Cadence simulators. For more information on the SimVision user interface, refer to the Cadence *SimVision Analysis Environment User Guide*.

In interactive simulation, you can cross-select design objects between SimVision tools and Virtuoso Schematic Editor. For details, see "Cross-Selecting Objects during Interactive Simulation" on page 85.

If you want to view the simulation results output in the SimVision window, then the SKILL variable *simNCVerilogNostdout* should be set to `nil` in the CIW or `.simrc` before running the simulation. The default value of this variable is `t`.

Note: In the SimVision user interface there is a command called *Reinvoke* in the File menu. When you enter the SimVision interface from NC-Verilog Integration Environment this option will not work.

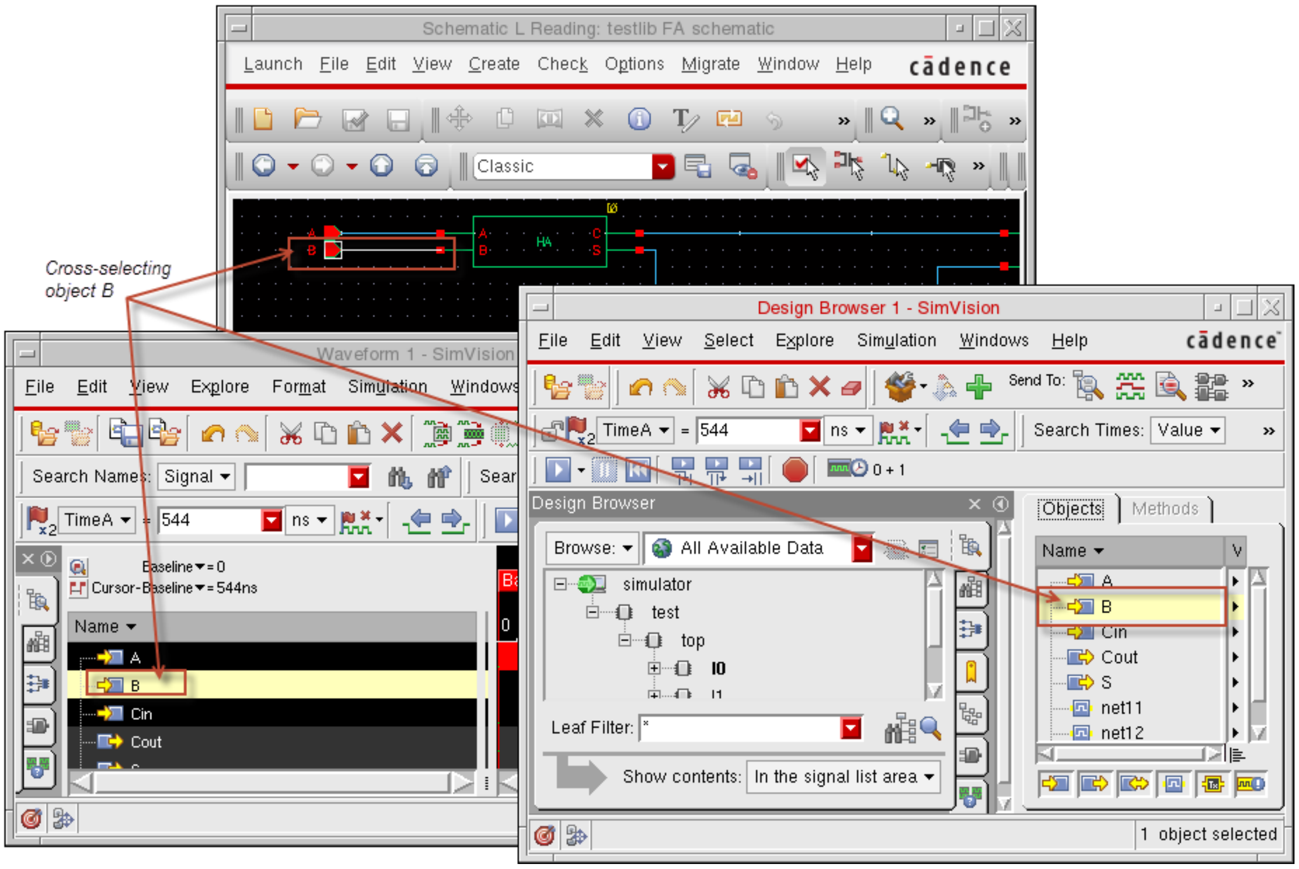
Cross-Selecting Design Objects

You can cross-select the objects of a design between Virtuoso Schematic Editor and the SimVision tools. Cross-selecting objects helps you identify a specific object in the schematic design corresponding to the object in SimVision, and vice versa.

You can cross-select objects when you perform interactive simulation. You can also use the available simulation data of a design and cross-select the design objects. The simulation data can be generated using the NC-Verilog Integration Environment or a standalone Verilog simulator. You can also cross-select iterated instances.

The following figure illustrates how you can cross-select an object.

Cross-Selecting an Object



This section includes the following topics:

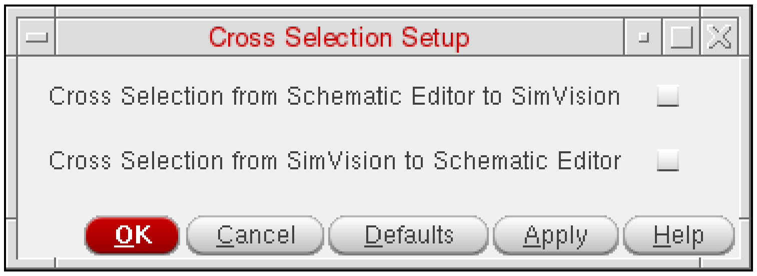
- Cross-Selecting Objects during Interactive Simulation
- Cross-Selecting Objects Using Simulation Data.
- Cross-Selecting Iterated Instances

Cross-Selecting Objects during Interactive Simulation

You can cross-select the objects of a design between SimVision tools and Virtuoso Schematic Editor when you perform interactive simulation. For information on interactive simulation, see “Introduction” on page 75 and “Simulating a Design” on page 83.

To cross-select objects during interactive simulation:

1. Choose *Setup — Cross Selection* from the main NC-Verilog window. The Cross Selection Setup form appears.



Cross Selection Setup

Cross Selection from Schematic Editor to SimVision ☐

Cross Selection from SimVision to Schematic Editor ☐

2. Select *Cross Selection from Schematic Editor to SimVision* to be able to select an object in the schematic design and have the corresponding simulation object selected in the SimVision user interface.
3. Select *Cross Selection from SimVision to Schematic Editor* to be able to select a simulation object in SimVision and have the corresponding object selected in the schematic design.
4. Click *OK*.

When SimVision is launched for interactive simulation, you can cross-select objects depending on the cross-selection setup. You can cross-select objects between Virtuoso Schematic Editor and SimVision tools, including SimVision Design Browser, SimVision Waveform, and SimVision Tracer. See the figure Cross-Selecting an Object.

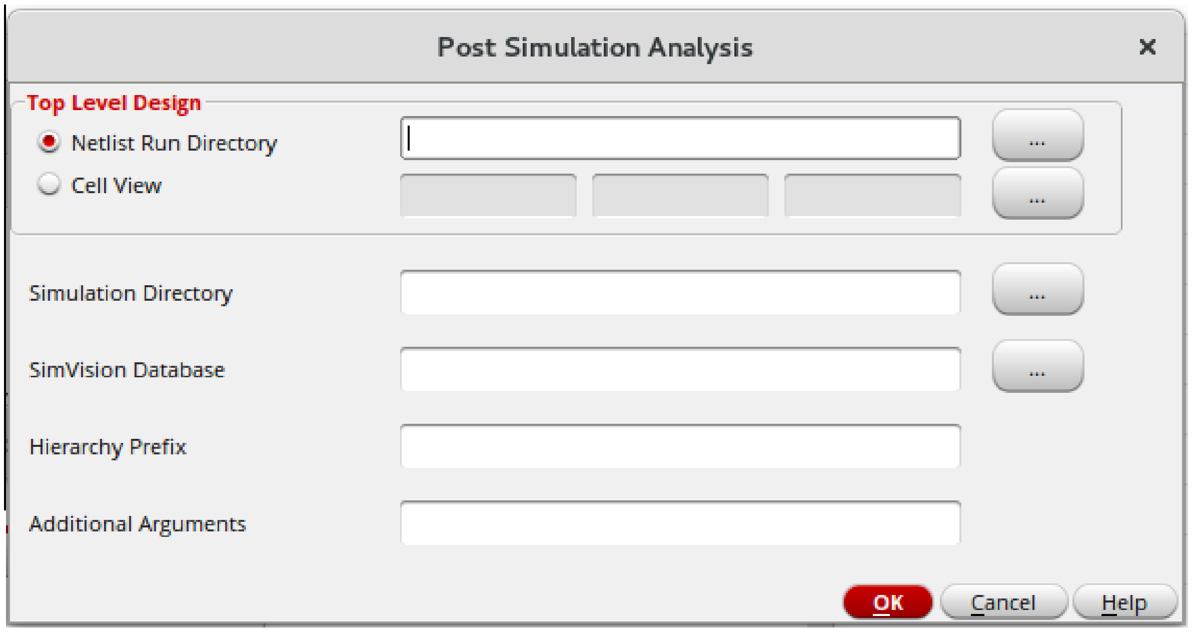
Cross-Selecting Objects Using Simulation Data

You can cross-select the objects of a design between SimVision tools and Virtuoso Schematic Editor using the available simulation data to perform post-simulation analysis. The simulation data can be generated by the NC-Verilog Integration Environment or a standalone Verilog simulator. For example, you can generate the Verilog netlist of a design using the NC-Verilog Integration Environment. You can then use this netlist to simulate the design using the `irun` tool from the command line. The results of this simulation can be used to cross-select design objects between SimVision tools and Virtuoso Schematic Editor.

The executable and log file names will depend on the simulator being used. For the changes in the executable and log file name when using the Xcelium simulator, see Running Simulations with Xcelium.

To specify the simulation data for cross-selecting design objects:

1. Choose *Commands — Post Simulation Analysis* from the main NC-Verilog window. The Post Simulation Analysis form appears.



2. Specify the following information in the form. You can click the browse button next to a field to select the input data:

Field	Description
Netlist Run Directory	Select this option if you want to specify the run directory. Then specify the path of the run directory.
Cell View	Select this option if you want to specify the top-level design cellview instead of the run directory. Then specify the library, cell, and view.
Simulation Directory	Specify the path of the simulation database.
SimVision Database	Specify the path and filename of the SimVision database `.trn` file.
Hierarchy Prefix	This field includes the hierarchy prefix used in the testbench, such as `test.top` or `test:top`. You can use a period (.) or a colon (`:`) as the delimiter in the hierarchy prefix.

Field	Description
Additional Arguments	This field lets you specify additional arguments that must be passed on to xrun. You must specify the arguments as follows: `-snapshot incr -xmlibdirname ./incrsnap`

3. Click *OK*.

   The specified SimVision database opens in SimVision Design Browser and you can cross-select objects.

   Note: When you initiate the cross-selecting functionality for post-simulation analysis, the cross-selection options in the Cross Selection Setup form are selected automatically.

To cross-select an object, select it in SimVision Design Browser. The schematic with the corresponding object gets highlighted in Virtuoso Schematic Editor. You can cross-select objects between Virtuoso Schematic Editor and SimVision tools, including SimVision Design Browser, SimVision Waveform, and SimVision Tracer. See the figure Cross-Selecting an Object.

Cross-Selecting Iterated Instances

You can cross-select the iterated instances in a design during interactive simulation or post-simulation analysis.

To enable cross-selection of iterated instances:

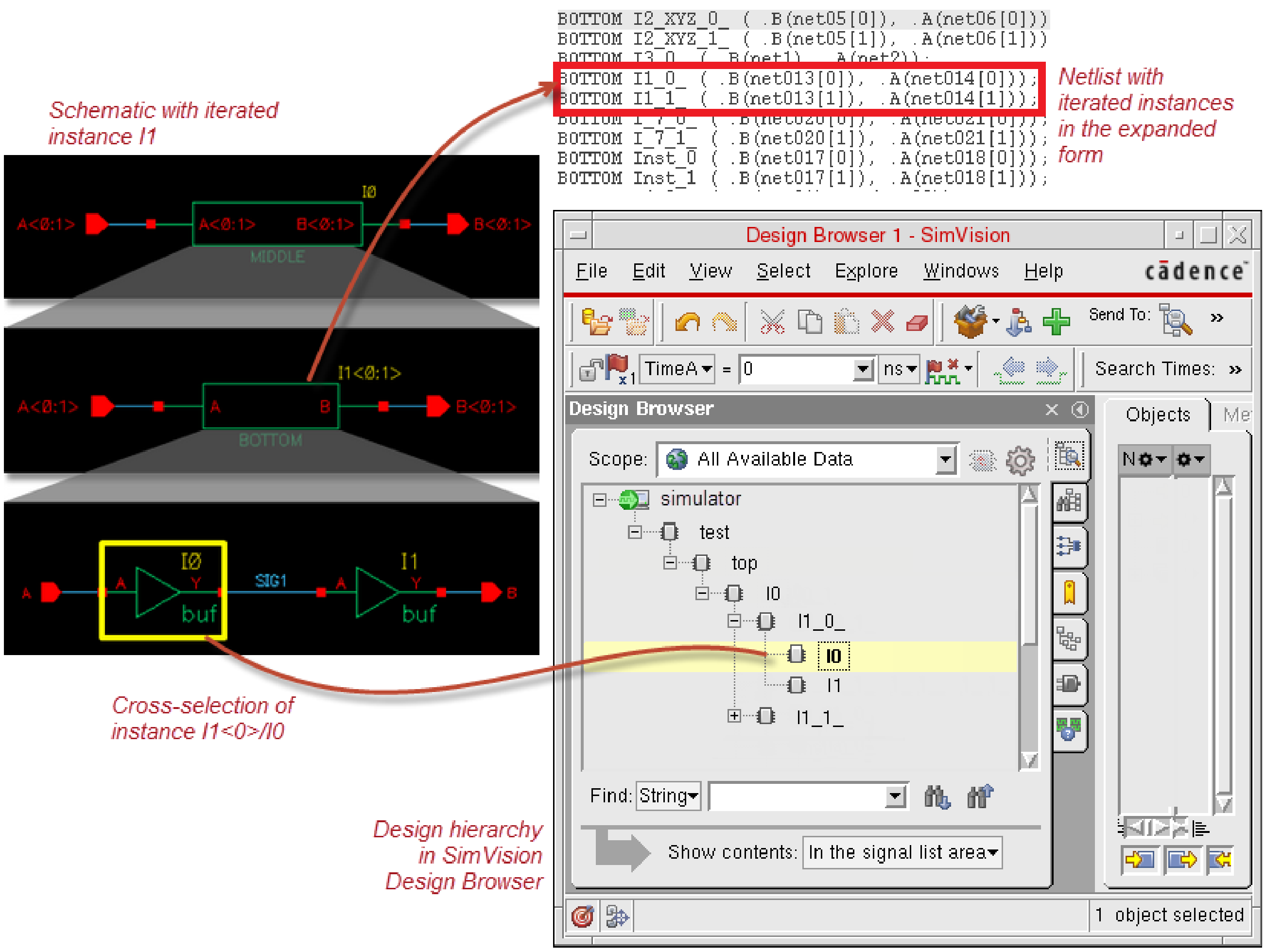
1. Ensure that the iterated instances are printed in the expanded form in the netlist in the format `baseName_bit_`, such as `I0_0_`.

   For this, set the flag `vlogExpandIteratedInst` to `t` in Virtuoso CIW or in `.simrc` before netlisting the design.

2. Set the flag to enable cross-selection of iterated instances `vlogProbeExpandedIterInst` to `t` in Virtuoso CIW or in `.simrc`.

For cross-selection, the expanded names of the iterated instances in the netlist are mapped to their corresponding instances in the schematic. For example, the expanded iterated instance `I1_0_` in the netlist is mapped to `I1<0>` in the schematic. When you select `I1_0_` in SimVision, the corresponding instance `I1<0>` is selected in Virtuoso Schematic Editor automatically.

The following figure illustrates the iterated instance `I1<0:1>` in the schematic view and its corresponding netlist, where the iterated instances are printed in the expanded form as `I1_0_` and `I1_1_`. The figure also shows cross-selection of the iterated instance.



```
BOTTOM I2_XYZ_0_  ( .B(net05[0]), .A(net06[0]))
BOTTOM I2_XYZ_1_  ( .B(net05[1]), .A(net06[1]))
BOTTOM I3_0_  ( .B(net1), .A(net2));
BOTTOM I1_0_  ( .B(net013[0]), .A(net014[0]));
BOTTOM I1_1_  ( .B(net013[1]), .A(net014[1]));
BOTTOM I_7_0_  ( .B(net020[0]), .A(net021[0]));
BOTTOM I_7_1_  ( .B(net020[1]), .A(net021[1]));
BOTTOM Inst_0  ( .B(net017[0]), .A(net018[0]));
BOTTOM Inst_1  ( .B(net017[1]), .A(net018[1]));
```

Netlist with iterated instances in the expanded form
Schematic with iterated instance I1
A<0:1>
I0
A<0:1>
B<0:1>
B<0:1>
MIDDLE
I1<0:1>
A<0:1>
A
B
B<0:1>
BOTTOM
A
I0
A
Y
buf
SIG1
I1
A
Y
buf
B
Cross-selection of instance I1<0>/I0
Design hierarchy in SimVision Design Browser
Design Browser 1 - SimVision
File Edit View Select Explore Windows Help
cādence
Send To:
TimeA = 0 ns
Search Times:
Design Browser
Objects
Scope: All Available Data
simulator
test
top
I0
I1_0_
I0
I1
I1_1_
Find: String
Show contents: In the signal list area
1 object selected

Notes:

- To avoid conflicts during cross-selection of iterated instances, ensure that the scalar instances in the schematic are not named in the format *`baseName_digits_`*, such as `I0_0_`. Moreover, the schematic should not have illegal Verilog names that require mapping to other legal Verilog names.
- If an iterated instance is selected in the schematic, its corresponding scalar instance in SimVision is not selected automatically because of ambiguity in the names.

For more information on iterated instances, see "Iterated Instances Support" on page 109.

Running a Batch Simulation

Running a batch simulation results in the execution of all the three options: Compile, Elaborate and Simulate in one go.

You can start batch simulation by selecting the Batch option in the main NC-Verilog window. The batch option is part of the Simulate Options in the main NC-Verilog window.

When a batch simulation finishes processing, a dialog box appears. The dialog box informs you that the batch job in the specified run directory completed successfully.

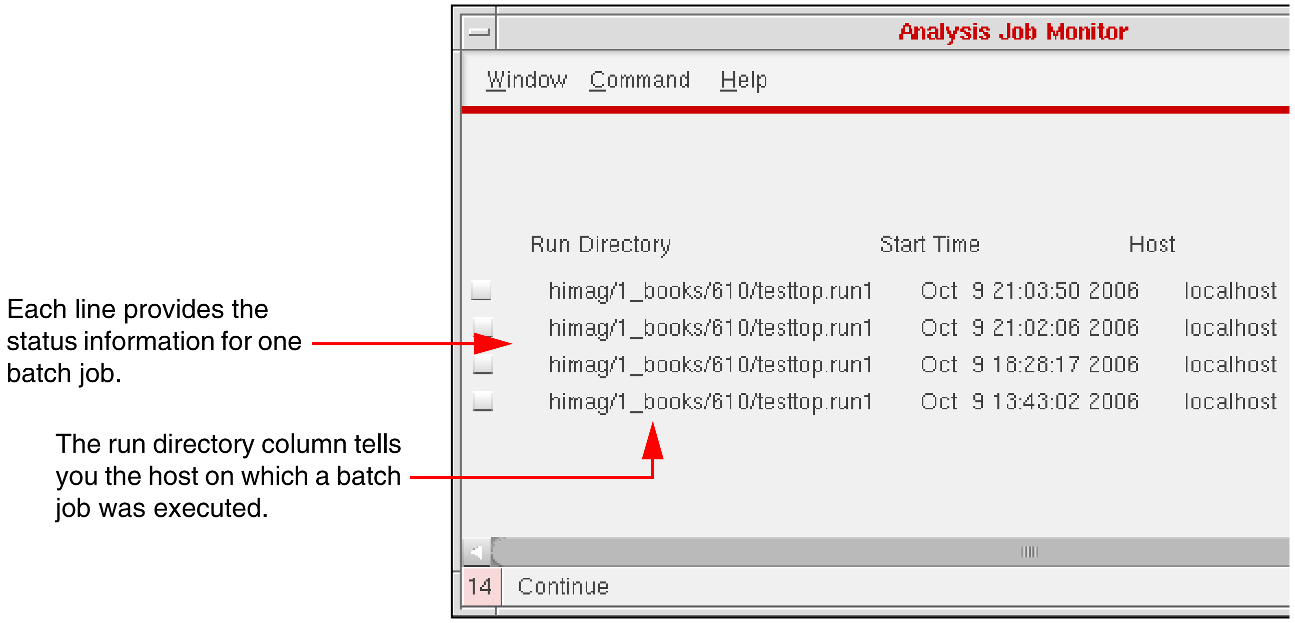
Monitoring Batch Simulations

The NC-Verilog Integration Environment lets you suspend batch jobs temporarily so that no processing time is allocated to the job. You can continue a suspended batch simulation whenever you choose.

To suspend or continue a batch simulation, you use the commands and options available on the Analysis Job Monitor form.

You can also change the priority of a batch job so that more or less processing time is allocated to the job.

To open the Job Monitor form, you select the Job Monitor command from the Results Menu in the main NC-Verilog window. The Analysis Job Monitor form appears.


Analysis Job Monitor
Window
Command
Help
Each line provides the status information for one batch job.
The run directory column tells you the host on which a batch job was executed.
14
Continue

Run Directory	Start Time	Host
himag/1_books/610/testtop.run1	Oct 9 21:03:50 2006	localhost
himag/1_books/610/testtop.run1	Oct 9 21:02:06 2006	localhost
himag/1_books/610/testtop.run1	Oct 9 18:28:17 2006	localhost
himag/1_books/610/testtop.run1	Oct 9 13:43:02 2006	localhost

The Analysis Job Monitor form displays the status for each batch job currently running or which has run during the current Schematic Editor session.

For more information on Job Monitor, see Job Monitor on page 60.

Comparing Simulation Results

SimCompare is used to compare the results obtained from different simulations. It provides a text description of any differences found.

Before using the tool, you need to set the simulation comparison options in the Simulation Comparison Setup form. This form can be accessed by selecting the Simulation Compare command from the Setup Menu in the main NC-Verilog window. For more information on the Simulation Comparison Setup form, refer to Simulation Compare on page 53.

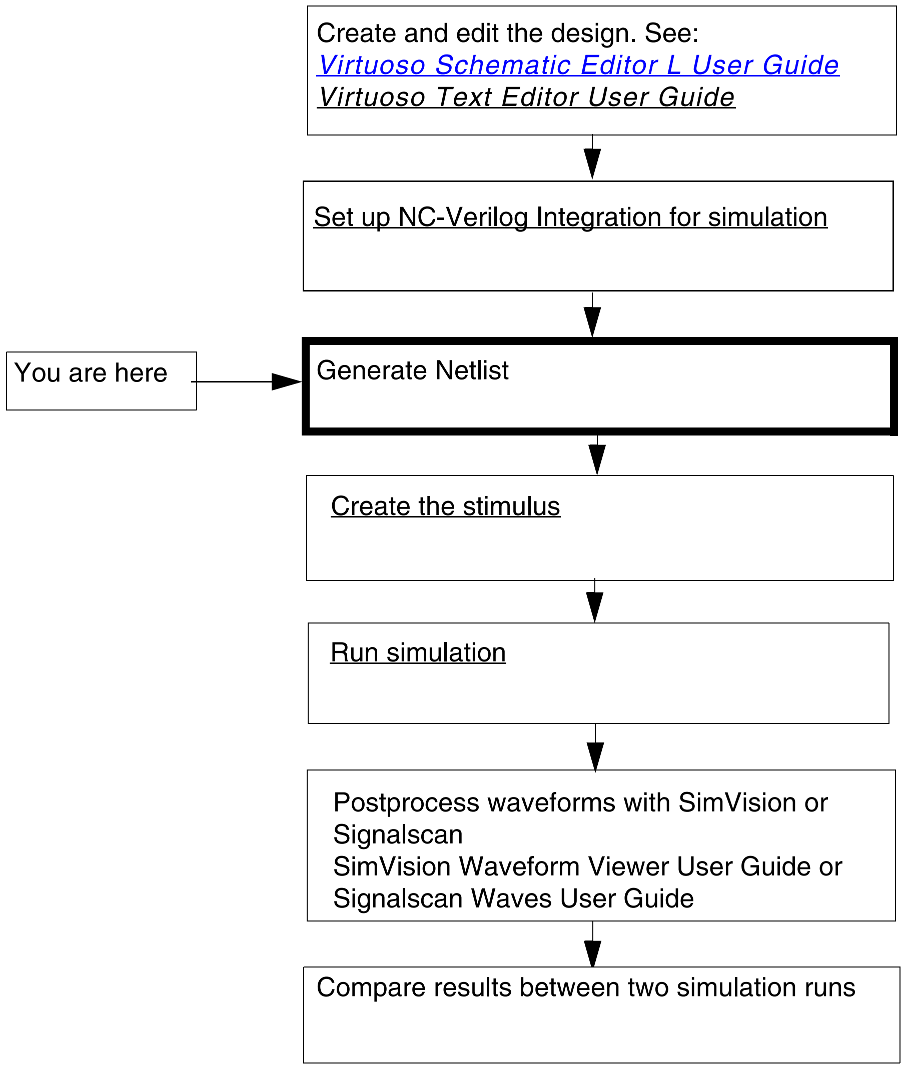
To start the comparison of simulation result, select the Simulation Compare command from the Fixed Menu. Alternatively you can also select the Simulation Compare command from the Commands menu. On selection of Simulation Compare command, SimCompare is launched in the background. For more information on using the SimCompare tool, refer to the *SimCompare User Guide*.

5

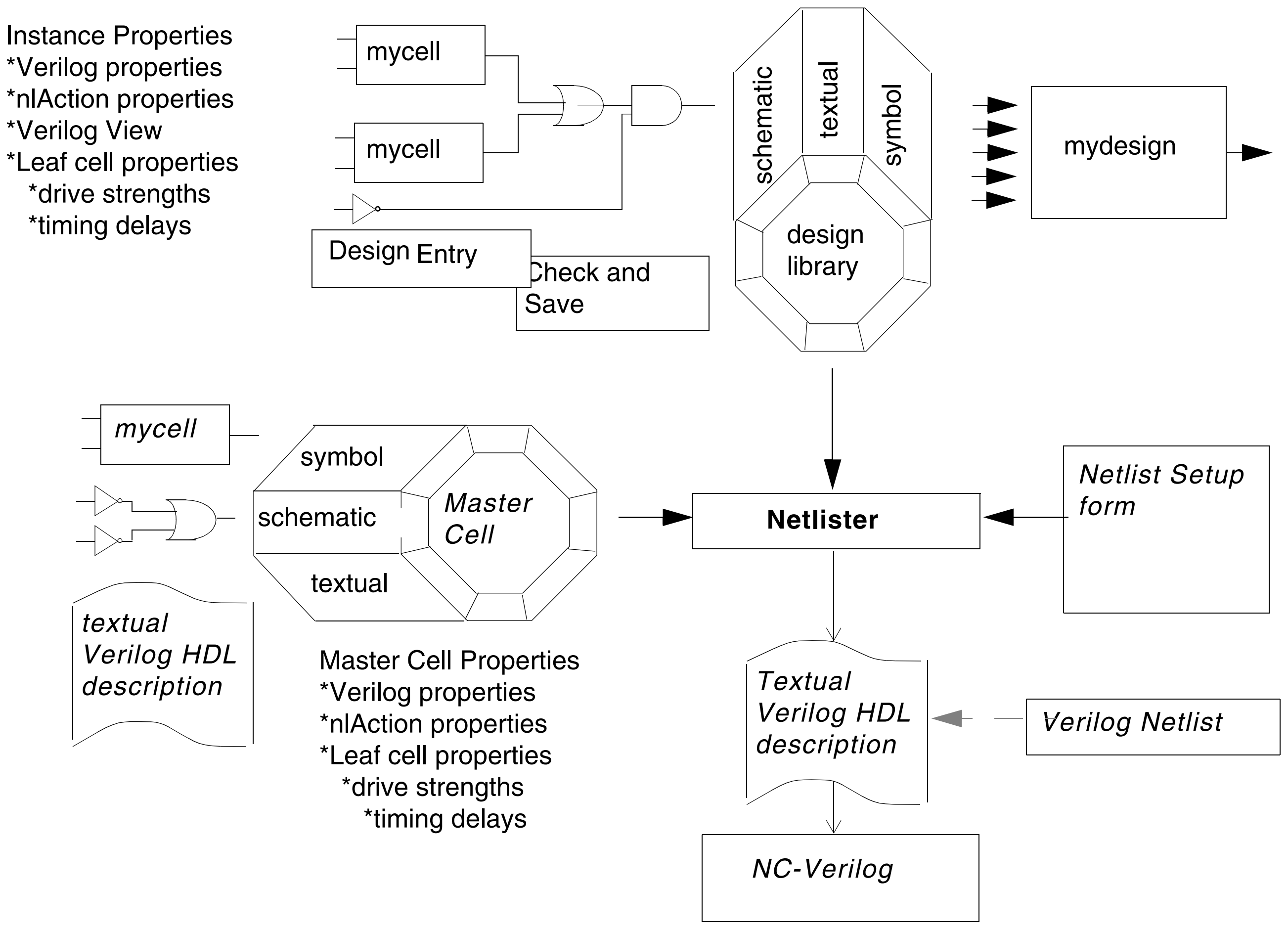
Netlisting

When you netlist your design, the netlister reads from the top level schematic or the HDL textual view of your design, checking for instance names, connectivity, any properties you have given to instantiations of cells, and properties that you assign to instantiations that take precedence over the properties assigned to the master cells. The result is a textual description of your design and a test fixture template. This chapter describes the following:

Simulation Process Flowchart


Create and edit the design. See:
Virtuoso Schematic Editor L User Guide
Virtuoso Text Editor User Guide
Set up NC-Verilog Integration for simulation
You are here
Generate Netlist
Create the stimulus
Run simulation
Postprocess waveforms with SimVision or Signalscan
SimVision Waveform Viewer User Guide or Signalscan Waves User Guide
Compare results between two simulation runs

Netlister Inputs and Outputs



Property Types

There are four property types that you can assign to the schematic views of master cells and instantiations of their symbol views. Those property types are:

- Verilog properties
- Timing delays and drive strengths
- Verilog views
- Netlister action (*nlAction*) properties

Note: With the exception of the netlister action (*nlAction*) properties, you must check and save a schematic if you want any of these property type values to be written as part of the schematic view that the netlister uses. The advantage of not having to save netlister action properties is that the netlister will temporarily ignore instances during netlisting.

Note: In addition to properties assigned to the schematic view, you can also assign the `lxRemoveDevice` property at the instance level. This property is used to short terminals of a device. For more details, refer to the Removing Devices with Multiple Terminals section of *Open Simulation System Reference*.

Verilog Properties

Verilog® properties are translated to

- ❑ parameter statements when you place them on master cells
- ❑ defparam statements that redefine parameter values when you place them on instances

Timing Delays and Drive Strengths

You can assign timing delays and drive strengths to leaf cells (that is, the cells that contain no instantiations).

Verilog Views

Verilog view types identify simulation models that the netlister references when netlisting individual instances in your design.

In addition to the cellviews listed in your view list, you can select the following types of simulation models:

- LOGIC Automation Incorporated (LAI) models
- Logic Modeling Systems Incorporated (LMSI)

LAI provides libraries of software models used for board-level devices. The LAI models are linked to the NC-Verilog® simulator through the LAI interface software provided with NC-Verilog.

LMSI produces hardware modelers and the model and interface software that supports these modelers. Hardware modeling enables the use of silicon chips to model the behavior of board-level devices. The LMSI hardware models are connected to the NC-Verilog simulator through interface software based on the Programming Language Interface (PLI).

Netlister Actions

Netlister action (*nlAction*) properties direct the netlister to perform specific actions. For example, you can assign an *ignore* netlister action property to instances of discreet components to prevent them from being netlisted.

The valid netlister actions are *ignore* and *stop*. The syntax is

```
nlAction="ignore"
nlAction="stop"
```

Inherited Connection Support

Inherited connections is an extension to the connectivity model that allows you to create global signals and override their names for selected branches of the design hierarchy. This flexibility allows you to use

- Multiple power supplies in a design
- Overridable substrate connections
- Parameterized power and ground symbols

This section explains how the Verilog netlister handles multiple power and ground supplies in a Cadence Virtuoso® Design Environment schematic design.

Note: You can use the `hnlUserStubCVList` variable to specify the stub cellviews, with the option to print the stub cellviews along with the interface details to resolve any inherited connections or power and ground net expressions.

To learn about connectivity and naming conventions for inherited connections and how to add and edit net expressions in a schematic or symbol cellview, refer to the *Virtuoso Schematic Editor L User Guide.*

- Support features
- Limitations
- Design example
- Netlist example

Features

Support for inherited connections in the NC-Verilog Integration Environment includes

- **Pseudo ports** The Verilog language does not support global signals. To support inherited connections, pseudo ports are generated through the hierarchy from where the net expression property is set to where the net expression is defined. One pseudo port is created for each inherited signal.
- **Netlisting Modes** The Verilog HNL netlister supports inherited connections in both full and incremental netlisting modes.

Limitations

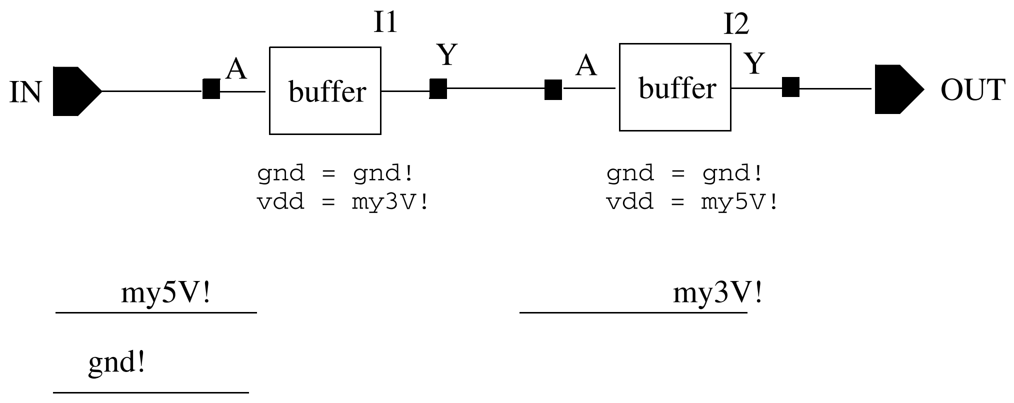
- Due to the introduction of pseudo ports in the resulting netlist, inherited connections cannot be inherited through text views. This is because the netlister cannot insert ports into a user-written textual description.
- Because the netlister creates a new pseudo port for each inherited terminal by default, the resulting netlist will not absolutely match the netlist for the functional view, which is based on the module's exact pin count. Consequently, simulation comparison tools, such as SimCompare, may report simulation differences due to the extra pseudo ports. Also, when backannotation is performed on a lower level-module that has extra pseudo ports, the SDF annotator of the Verilog simulator will flag a backannotation error.

  Note: When backannotation is performed on the top-level module, no error flags are generated because the components in the SDF are primitives.

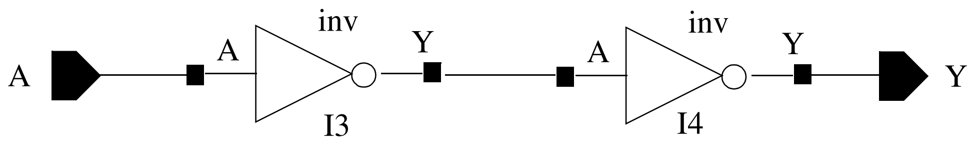
Design Example

This section describes how you create an inherited property in VSE that handles multiple supplies.

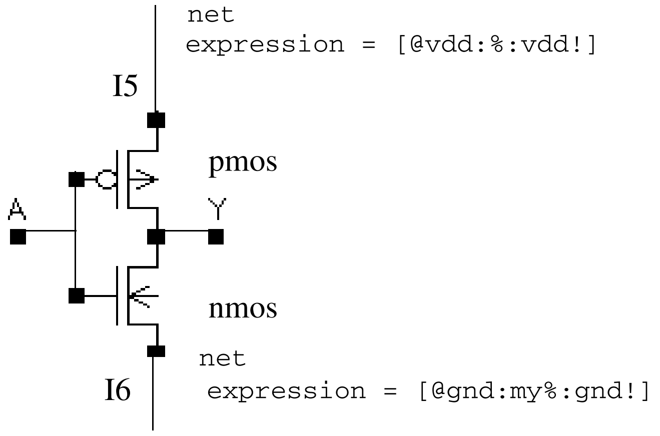
Schematic diagram for library: *myLib*, **cell:** *top*, **and view:** *schematic*



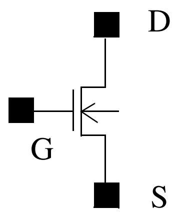
Schematic diagram for library: *myLib*, **cell:** *buffer*, **and view:** *schematic*



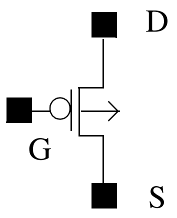
Schematic diagram for library: *myLib*, **cell:** *inv*, **and view:** *schematic*



Schematic diagram for library: *myLib*, **cell:** *nmos*, **and view:** *symbol*



Schematic diagram for library: *myLib*, **cell:** *pmos*, **and view:** *symbol*
l



The example assumes a need to create a generic design component, such as an inverter.

Depending on the block the inverter is instantiated in, the connections to power may be either 5 volt or 3 volt. This information cannot be inherently placed within the inverter schematic because the information must be common. Accordingly, the schematic uses the generic net

- `vdd!` for power
- `gnd!` for ground

Now, consider the user instantiating these components (or blocks built out of these components) at some arbitrary level of hierarchy above, who needs to redefine `vdd!` to mean `my5V` and `my3V` respectively.

In the schematic view of the cell *inv*, instead of using the general power and ground symbols, nets have been connected to those two nodes, and labels attached to them allowing inherited redefinition of the net name. The power connection (`vdd!`) has been defined to be the expression `[@vdd:%:vdd!],` which is interpreted as "set the name of this net to the name assigned to the property `vdd`, if you find it; otherwise, the default name of this net should be `vdd!`." The default net must be global.

In the schematic view of the cell *inv*, the net expression for the ground connection (`gnd!`) is `[@gnd:my%:gnd!]`. This means 'find the value for the property `gnd`, prepend the prefix `my`, and use the result as the inherited net name'. The result is: `mygnd!` The net `mygnd!` must exist where the property `gnd` is set.

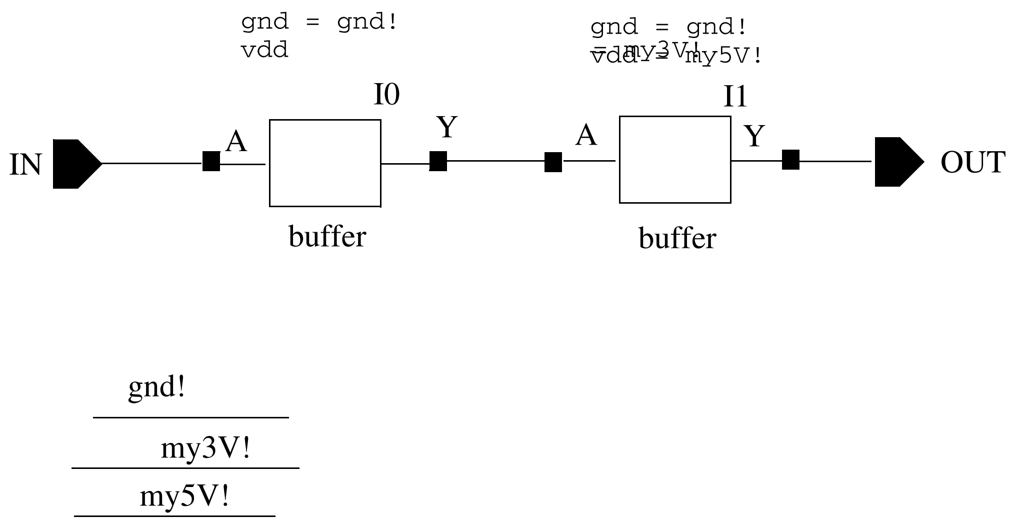
In the top-level schematic, the properties `gnd` and `vdd` have been added to instance `I1` with values `gnd!` and `my3V` respectively. In doing so, we have redefined the two parameterized nets in the *inv* schematic, such that the top net will now be replaced by the global net `my3V` and the bottom net will be replaced by the global net `mygnd!`

Similarly by setting the properties `gnd` and `vdd` to `gnd!` and `my5V` respectively in instance `I2`, we redefine the topmost net in the inverter schematics referenced by instance `/I2/I3` and `/I2/I4` to be the net `my5V`. The bottom net is `mygnd!`

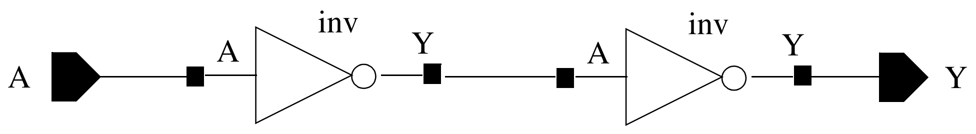
Netlist Example

This section describes how the netlister handles inherited connections within the NC-Verilog Integration Environment.

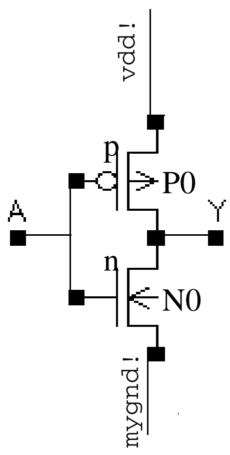
Schematic diagram for library: *mylib*, **cell:** *top*, **and view:** *schematic*



Schematic diagram for library: *mylib*, **cell:** *buffer*, **and view:** *schematic*



Schematic diagram for library: *mylib*, **cell:** *inv*, **and view:** *schematic*



Because the Verilog language does not support global nets, OSS offers the HNL Verilog formatter a special way to support inherited connection within the NC-Verilog Integration Environment. Additional ports are generated down the hierarchy from where the net expression property is set to where the net expression is defined. One additional port is created for each inherited net.

The resulting netlist will look something like the following for the above example:

```
// Global nets module

`celldefine
module cds_globals;
   supply1 my3V_;
   supply0 gnd_;
   supply1 my5V_;
endmodule
`endcelldefine
// Library - mylib, Cell - myinv, View - schematic
// LAST TIME SAVED: Aug 13 15:16:34 1998
// NETLIST TIME: Aug 13 15:16:41 1998
`timescale 1ns / 1ns

module myinv ( Y, inh_gnd, inh_vdd, A );
   output  Y;
   input  A;
   inout  inh_gnd, inh_vdd;

   specify
      specparam CDS_LIBNAME  = "mylib";
      specparam CDS_CELLNAME = "myinv";
      specparam CDS_VIEWNAME = "schematic";
   endspecify

   nmos N0( Y, inh_gnd, A);
   pmos P0( inh_vdd, Y, A);
```

```
endmodule

// Library - mylib, Cell - buffer, View - schematic
// LAST TIME SAVED: Aug 13 15:09:32 1998
// NETLIST TIME: Aug 13 15:16:41 1998
`timescale 1ns / 1ns

module buffer ( Y, inh_gnd, inh_vdd, A );
   output  Y;
   input  A;
   inout  inh_gnd, inh_vdd;

   specify
      specparam CDS_LIBNAME  = "mylib";
      specparam CDS_CELLNAME = "buffer";
      specparam CDS_VIEWNAME = "schematic";
   endspecify

   myinv I3 ( Y, inh_gnd, inh_vdd, net6);
   myinv I2 ( net6, inh_gnd, inh_vdd, A);
endmodule

// Library - mylib, Cell - top, View - schematic
// LAST TIME SAVED: Aug 13 15:09:46 1998
// NETLIST TIME: Aug 13 15:16:41 1998
`timescale 1ns / 1ns
module top ( OUT, IN );
   output  OUT;
   input  IN;

   specify
      specparam CDS_LIBNAME  = "mylib";
      specparam CDS_CELLNAME = "top";
      specparam CDS_VIEWNAME = "schematic";
   endspecify

   buffer I0 ( net4, cds_globals.gnd_, cds_globals.my3V_, IN);
   buffer I1 ( OUT, cds_globals.gnd_, cds_globals.my5V_, net4);
endmodule
```

Enhanced Support for Inherited Connections

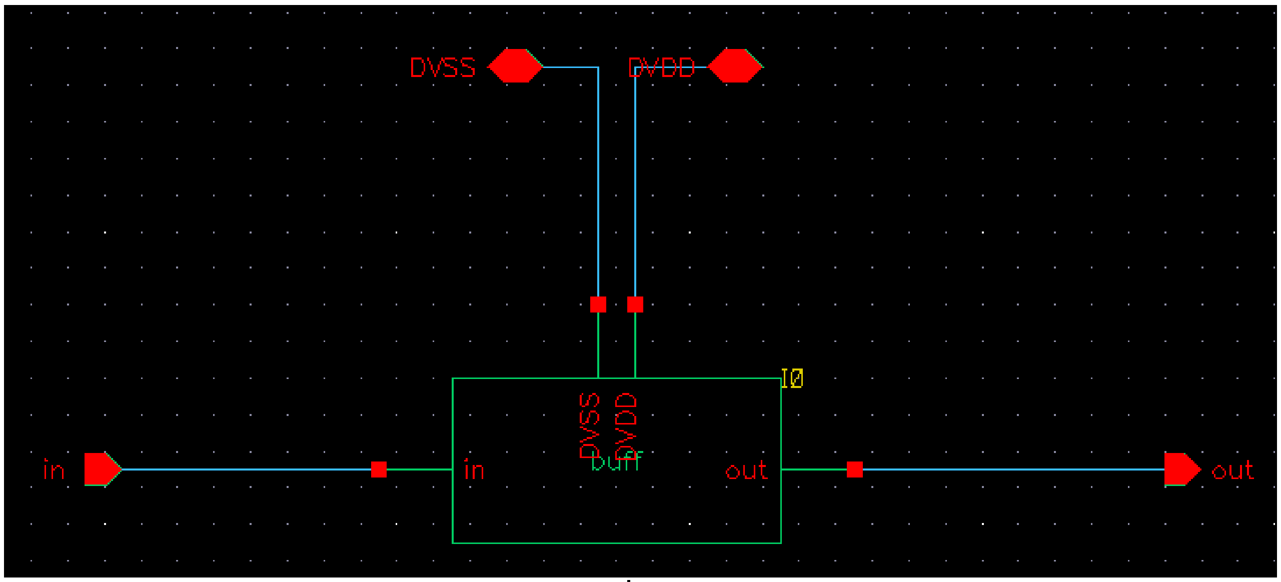
A new SKILL environment variable, `simPrintInhConnAttributes`, allows you to prevent the creation of pseudo ports and get the inherited connections information in the netlists which you started off with at the time of design entry.

`simPrintInhConnAttributes` is a boolean variable with a default value as `nil`. You can set this environment variable in the `.simrc` file. The syntax is as follows:

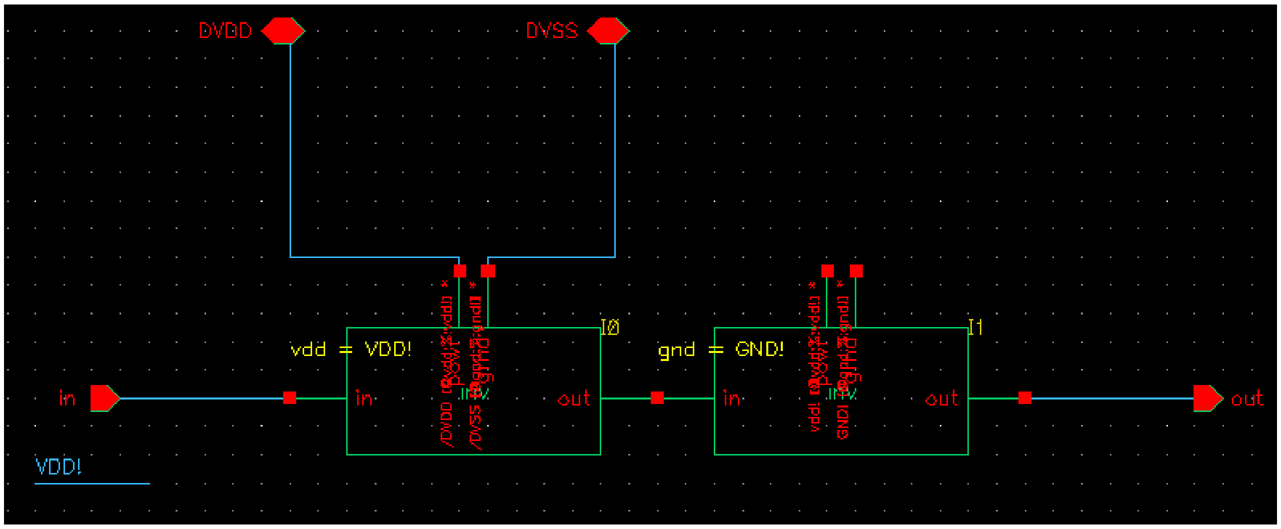
```
simPrintInhConnAttributes = t/nil
```

As the default value is `nil`, by default the Verilog netlister creates pseudo terminals for intermediate levels of the hierarchy.

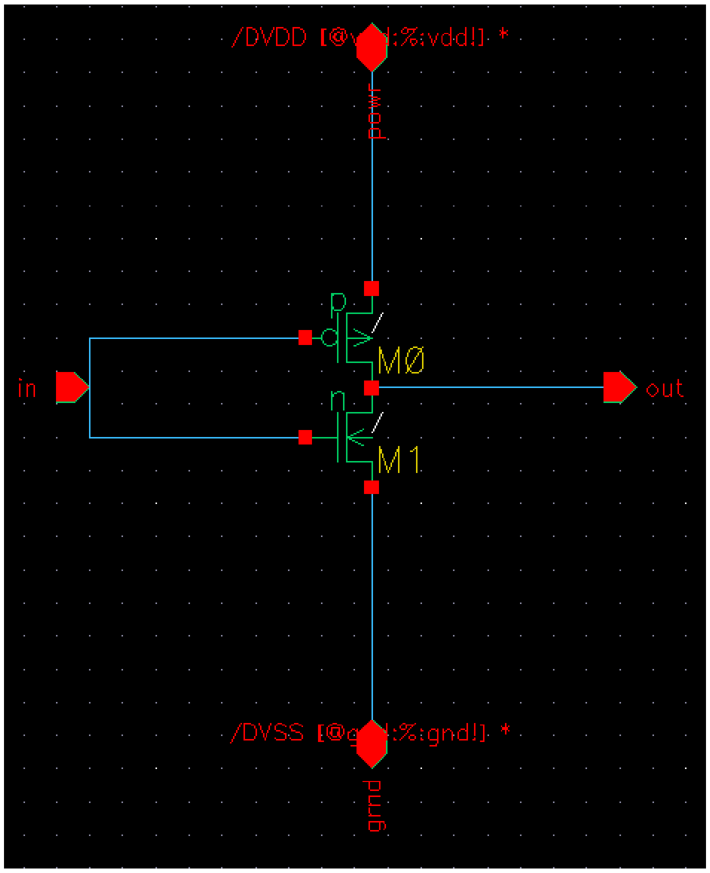
The following figure shows an example of a buffer that uses explicit terminals with inherited connections for the power and the ground supplies.



Design-Hierarchy-Descend Read...



Design-Hierarchy-Descend Read...



For the above example, there are two versions of the netlist generated by Verilog Netlister corresponding to:

1. `simPrintInhConnAttributes = 'nil`
2. `simPrintInhConnAttributes = 't`

The Verilog netlist obtained when `simPrintInhConnAttributes = 'nil` is as follows. Notice that in each of the netlists, the inherited connections are translated to pseudo ports highlighted in bold.

For cds0

```
// Library - test2, Cell - inv, View - schematic
// LAST TIME SAVED: Mar 24 15:37:26 2004
// NETLIST TIME: Jun 29 14:52:42 2005
`timescale 1ns / 1ns

module inv ( out, .grnd(inh_gnd), .powr(inh_vdd), in );
output  out;
```

```
inout  inh_gnd, inh_vdd;

input  in;

specify
    specparam CDS_LIBNAME  = "test2";
    specparam CDS_CELLNAME = "inv";
    specparam CDS_VIEWNAME = "schematic";
endspecify

tranif1 M1( out, inh_gnd, in);
tranif0 M0( inh_vdd, out, in);
endmodule
```

For cds1

```
// Library - test2, Cell - buff, View - schematic
// LAST TIME SAVED: Oct  6 12:04:54 2004
// NETLIST TIME: Jun 29 14:52:42 2005
`timescale 1ns / 1ns

module buff ( out, DVDD, DVSS, inh_vdd, in );
output  out;

inout  DVDD, DVSS;

input  in;

inout  inh_vdd;


specify
    specparam CDS_LIBNAME  = "test2";
    specparam CDS_CELLNAME = "buff";
    specparam CDS_VIEWNAME = "schematic";
endspecify

inv I1 ( .powr(inh_vdd), .grnd(cds_globals.GND_), .in(net6),
.out(out));
inv I0 ( .powr(DVDD), .grnd(DVSS), .in(in), .out(net6));

endmodule
```

For cds2

```
// Library - test2, Cell - top2, View - schematic
// LAST TIME SAVED: Mar 24 15:55:12 2004
// NETLIST TIME: Jun 29 14:52:42 2005
`timescale 1ns / 1ns

module top2 ( out, DVDD, DVSS, in );

output  out;

inout  DVDD, DVSS;
```

```
input  in;

specify
    specparam CDS_LIBNAME  = "test2";
    specparam CDS_CELLNAME = "top2";
    specparam CDS_VIEWNAME = "schematic";
endspecify

buff I0 ( .DVSS(DVSS), .DVDD(DVDD), .in(in), .out(out),
    .inh_vdd(cds_globals.vdd_));
endmodule
```

The Verilog netlist obtained when `simPrintInhConnAttributes = 't` is as follows. In the following netlists notice that the connectivity information is preserved and netExpression and netSet properties defined.

For cds0

```
// Library - test2, Cell - inv, View - schematic
// LAST TIME SAVED: Mar 24 15:37:26 2004
// NETLIST TIME: Oct 19 19:25:00 2004
`timescale 1ns / 1ns

module inv ( out, .grnd(gnd_), .powr(vdd_), in );

output  out;

(* netExpr = "gnd(gnd_)" *)inout  gnd_;
(* netExpr = "vdd(vdd_)" *)inout  vdd_;

input  in;


specify
    specparam CDS_LIBNAME  = "test2";
    specparam CDS_CELLNAME = "inv";
    specparam CDS_VIEWNAME = "schematic";
endspecify

tranif1 M1( out, gnd_, in);
tranif0 M0( vdd_, out, in);

endmodule
```

For cds1

```
// Library - test2, Cell - buff, View - schematic
// LAST TIME SAVED: Oct  6 12:04:54 2004
// NETLIST TIME: Oct 19 19:25:00 2004
`timescale 1ns / 1ns

module buff ( out, DVDD, DVSS, in );
supply1 VDD_;
supply0 GND_;

output  out;
```

```
inout  DVDD, DVSS;

input  in;

(* netExpr = "vdd(vdd_)" *) wire vdd_;

specify
    specparam CDS_LIBNAME  = "test2";
    specparam CDS_CELLNAME = "buff";
    specparam CDS_VIEWNAME = "schematic";
endspecify

(* netSet = "gnd" *)(* gnd = "GND_" *)inv I1 ( out, GND_, vdd_, net6);
(* netSet = "vdd" *)(* vdd = "VDD_" *)inv I0 ( net6, DVSS, DVDD, in);

endmodule
```

For cds2

```
// Library - test2, Cell - top2, View - schematic
// LAST TIME SAVED: Mar 24 15:55:12 2004
// NETLIST TIME: Oct 19 19:25:00 2004
`timescale 1ns / 1ns

module top2 ( out, DVDD, DVSS, in );

output  out;

inout  DVDD, DVSS;

input  in;

specify
    specparam CDS_LIBNAME  = "test2";
    specparam CDS_CELLNAME = "top2";
    specparam CDS_VIEWNAME = "schematic";
endspecify

buff I0 ( out, DVDD, DVSS, in);
endmodule
```

Using a Verilog File Reference

In the configuration view of a design, you can add a reference of an external Verilog file for binding a cell, an instance, or an occurrence to a view in the referenced file. For details, see *Referencing a Verilog File*.

If you have specified a Verilog file reference through Virtuoso Hierarchy Editor and then netlist the design using the NC-Verilog netlister, the netlister uses the referenced Verilog file for cell binding and instance binding.

Note: The NC-Verilog netlister does not support occurrence binding using an external Verilog file reference. It supports stop points only at the cell level but not at the instance and occurrence levels.

The netlister also creates a `config_file` file in the run directory that contains the binding information. Following is an example of `config_file`:

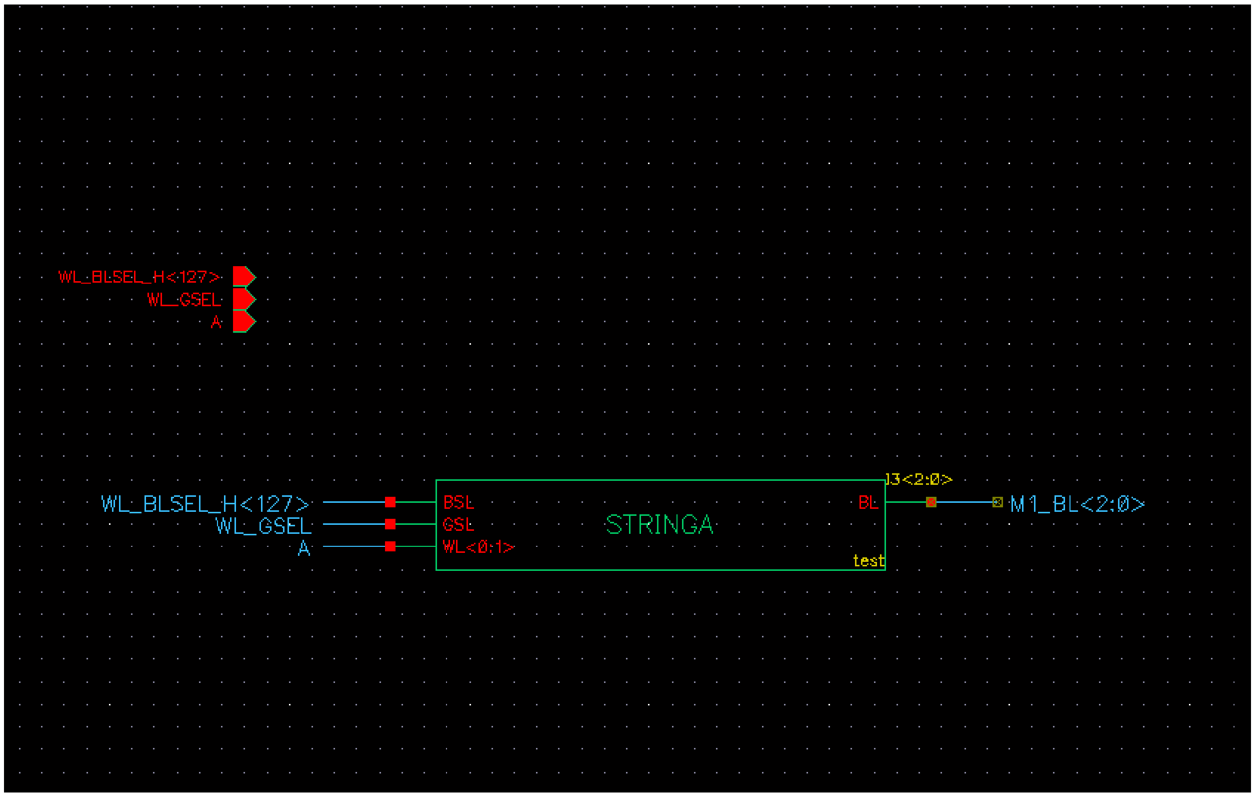
```
config cdns_cfg;

design worklib.test;

instance test.top.Iinvpp8_39 use tmplib1.inv ;

instance test.top.Iinvpp1_22 use tmplib1.inv ;

...

endconfig
```

Iterated Instances Support

NC-Verilog supports netlisting of iterated instances without any expansion according to the Verilog 2001 standard.

An iterated instance refers to an array of multiple instances. The iterated instance feature specifies that there is one-to-one mapping between a schematic design and a netlist with respect to iterated instances. This feature reduces netlist size, which further results in significant decrease in the time taken to generate a netlist. For example, in a Verilog design with an iterated instance of 8K range, a netlist is generated in approximately four minutes. This is an improvement over nine hours taken earlier without iterated instances support.

The following figure shows a design with iterated instance, I3<2:0>.



Earlier, each iterated instance was expanded to a list of individual instances and netlisted separately. The following example shows an iterated instance, `I3<2:0>`, which is expanded as three individual instances, `I3_2_`, `I3_1_`, and `I3_0_`:

```
STRINGA I3_2_(M1_BL[2], WL_BLSEL_H[127], WL_GSEL, {A,A});
STRINGA I3_1_(M1_BL[1], WL_BLSEL_H[127], WL_GSEL, {A,A});
STRINGA I3_0_(M1_BL[0], WL_BLSEL_H[127], WL_GSEL, {A,A});
```

The iterated instance netlisting feature is enabled by default. The following example shows an array of iterated instances, which is processed as a single instance, `I3<2:0>`:

```
STRINGA I3[2:0](M1_BL[2:0], WL_BLSEL_H[127], WL_GSEL, {A,A});
```

To disable the iterated instance netlisting, set the `vlogExpandIteratedInst` flag to true in the `.simrc` file or at CIW as:

```
vlogExpandIteratedInst = t
```

If a design module has multiple iterated instances with the same base name, the netlister prints them in the expanded form, even if the `logExpandIteratedInst` flag is not set to `t`. For example, consider a schematic that contains split iterated instances `I0<2:0>` and

`I0<3>`. In this case, the netlister prints the instances in the expanded form, as illustrated in the following Verilog netlist snippet:

```
    dummy I0_2_ ( cds_globals.gnd_, net3);
    dummy I0_1_ ( cds_globals.gnd_, net3);
    dummy I0_0_ ( cds_globals.gnd_, net3);
    dummy I0_3_ ( net2, cds_globals.gnd_);
```

If there are no aliases in a design, you can stop alias processing to further improve NC-Verilog performance. For example, if you set the `hnlIgnoreAlias` flag to true in a Verilog design with an iterated instance of 8K range, a netlist is generated in approximately 1 second. This is a further improvement over the 4 minutes taken when the flag value is false. The `hnlIgnoreAlias` flag is set to true as:

```
hnlIgnoreAlias = t
```

Controlling Netlister Actions

You can modify netlisting options at any time during a session by changing the options on the Netlist Setup form and then restarting the netlister.

The Netlist Setup form lets you

- Specify the order of precedence for cellview translation
- Specify the level of hierarchical expansion
- Specify the syntactical formats used in the netlist; for example, you can preserve bus structures [vectors] or translate them to bit netlists [scalars].
- Assign global nets to net types *supply0* and *supply1*
- Specify whether netlisting is performed in the foreground or background
- Specify how to synchronize terminals of an instance and its switched master

Using View Lists

The view list lets you specify the order of precedence that the netlister uses to select and netlist the cellviews for a given master cell. The following list shows the default order of precedence:

```
functional behavioral system hdl verilog schematic symbol
```

Beginning with the first cellview in the list, the netlister traverses the view list, in order, until an existing cellview is found. That cellview is then used to netlist the cell. This process is repeated until all the cells of your design are netlisted.

Note: In general, when you netlist a given cellview, you netlist the master cell property values. However, any property values that you assign to instantiations override the property values of the master cell.

Using Stop Lists

In addition to selecting a cellview for netlisting, you can control the hierarchical expansion of cells by specifying the stop list.

When the netlister encounters a cellview from the view list, the netlister checks to see if the cellview is in the stop list. If the cellview is in the stop list, the netlister stops further expansion of the design at this level. If the cellview is not in the stop list, the netlister continues the expansion by examining instances contained in the chosen cellview.

The order of cellviews in the stop list is irrelevant. The default stop list is

```
functional behavioral system hdl verilog symbol
```

You can use the Netlist Setup form to modify the view and stop lists.

Starting the Netlister

In the simulation environment, you must explicitly start the netlister

- Prior to simulating a new design
- When any cell in your design is modified
- When any cell instantiated from a source library is modified

To start the netlister, on the fixed menu click *Generate Netlist* or from the menu bar choose *Commands – Generate Netlist*.

User Messages

The netlister issues messages to inform you how netlisting is being performed. For example, it will let you know

- The syntactical formats you have selected to netlist; that is, whether you preserved bus structures (vectors) or translated them to bit netlists (scalars)

- Which view is used for netlisting each cell and cellviews at which hierarchical expansion is stopped
- When global nets are incorrectly specified
- If all design cells are being renetlisted or you are netlisting incrementally
- Any cellnames that are potentially illegal Verilog names and the names they are mapped to

You can set the `hnlVerilogVerboseLevel` variable to specify the type of messages that should be displayed on CIW. By default, all error, warning and info messages from netlister are displayed. Set this variable to `error` or `warning` to display only error or only error and warning messages, respectively.

Global Nets Not Power or Ground

You can specify that a net is a global signal by specifying an exclamation point (!) as the last character in its name in a schematic view.

You normally specify global signals in one of the following locations:

- The *Global Power Nets* or *Global Ground Nets* text entry fields
- The OSS global variable assignments to *simVerilogPwrNetList* or *simVerilogGndNetList*

When the netlister encounters global signal declarations in other locations, the netlister instantiates those signals in a special purpose module called `globals_module`.

The `globals_module` resides in the automatically generated `cds_global.v` file in the `hdlFilesDir` directory. The `globals_module` is instantiated in the test fixture file. The test fixture is the only place in which you can place values on these nets. Do not attempt to drive them in a design module or in the `globals_module` definition.

The signals instantiated in the `globals_module` module are of the type *wire*. In each design module, the netlister places continuous assignments that set the local values of the global nets to their global values. For example, if the netlister encountered the global signal `global_clock`, it would create the following module:

```
module globals_module;
wire global_clock;
endmodule
```

The netlister would also create the following continuous assignment in each design module:

```
assign global_clock=top.globals_inst.global_clock;
```

The netlister generates a test fixture template that includes an instantiation of the `globals_module` module. If you create your own test fixture template, you need to add an instantiation of the `globals_module` called `globals_inst` to your template.

Global signals on schematics are mapped to `global_`*`x`*, where *`x`* is a unique number. The mapping for a cell's global signals is contained in the `globalmap` file in the `ihnl` directory of the run directory. You can examine the `globalmap` file to see entries similar to the following that demonstrate the mappings:

```
gnd! global_0
vdd! global_1
global_clock! global3
```

To propagate signals on all the wires named `global_clock` in the design modules, assuming that `global_clock` maps to `global_3`, place lines similar to the following in the test fixture:

```
module test;

reg global_clock_value;
globals_module globals_inst();

assign
   top.globals_inst.global_3=global_clock_value;

initial
global_clock_value=0;
always
   begin
      #10;
      global_clock_value=~global_clock_value;
   end

endmodule;
```

Be sure to refer to global nets by their complete hierarchical names when you make assignments to them. Failing to refer to the complete hierarchical names can result in changes that have less than global scope.

Synchronizing Terminals

By default, the NC-Verilog netliser retains the design terminals during netlist generation. However, if there is a mismatch between the terminals of an instance/placed master and its switch master, the netlister expands the terminals in the netlist.

For all kinds of terminals, scalar, vector, or bundles, the netlister checks the names of terminals. If the names are precisely same, the netlister retains the bus terminals in the netlist. For example, if the placed master of an instance has terminals as a[0:3], b, and d and the switch master also has same terminals as a[0:3], b, and d, the netlist too has same terminals. In case of a mismatch, the netlister splits the terminals that mismatch. For example, if the placed master has terminals as a[0:2], b, and d and the switch master has terminals as a[0], a[1], a[2], b, and d, the netlister splits the terminal of the placed master as a[0], a[1], a[2].

However, you can choose to synchronize the mismatched terminals while generating a netlist using the *Terminal SyncUp* option on the *Netlist Setup* form. You can also set the `hnlVerilogTermSyncUp` variable in the `.vlogifrc` file.

The *Terminal SyncUp* option provides the following three choices:

- Expand on Mismatch: Retains the design terminals, but expands the mismatched terminals, as explained above. This is the default option.
- Honor Switch Master: Honors the terminals of the switch master and merges/expands the ports of the placed master to match with the switch master. This option does not work with explicit netlisting and is helpful when you want to netlist text views.
- Merge All: Merges the placed master and switch master terminals. The netlister creates pure buses or scalar terminals. Using this option, you can create pure explicit netlist for any type of design.

If there are terminals on the placed master or symbol view but not in the switched master or schematic view, by default, the netlist generation continues and the mismatched terminals are dropped from the netlist. This is because, by default, the `hnlVerilogTermMismatchAction` variable is set as `ignore`.

If you want to have an error or warning message for such terminals, set the `hnlVerilogTermMismatchAction` variable to any of the following values:

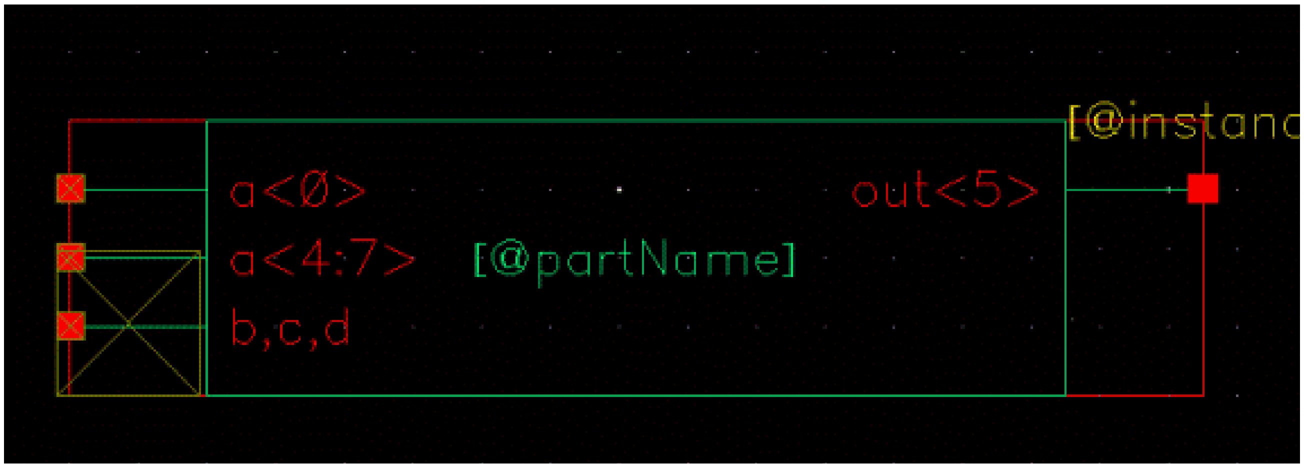
- `error`: Stops the netlist generation and prints an error message.
- `warning`: Continues netlist generation, but drops the mismatched terminal from the netlist and prints a warning message.

Note: Some terminals on placed masters are created with the `physOnly` attribute. These terminals are used to implement connectivity model and are by default ignored by an OSS-based flat netlister

Caution

If the** Terminal SyncUp **option is set as** `Honor Switch Master` **or** `Merge All`**, setting some netlist configuration options might give some unexpected results. For example, if you set the** `hnlVerilogNetlistNonStopCellExplicit` **option in** `.simrc` **or the** Drop Port Range **and** Preserve Buses **options in the** Netlist Setup **form and set the** Terminal SyncUp **option as** `Honor Switch Master`**, some unexpected results are generated. Therefore, it is recommended to set default values for the netlist configuration options while using Terminal SyncUp option.

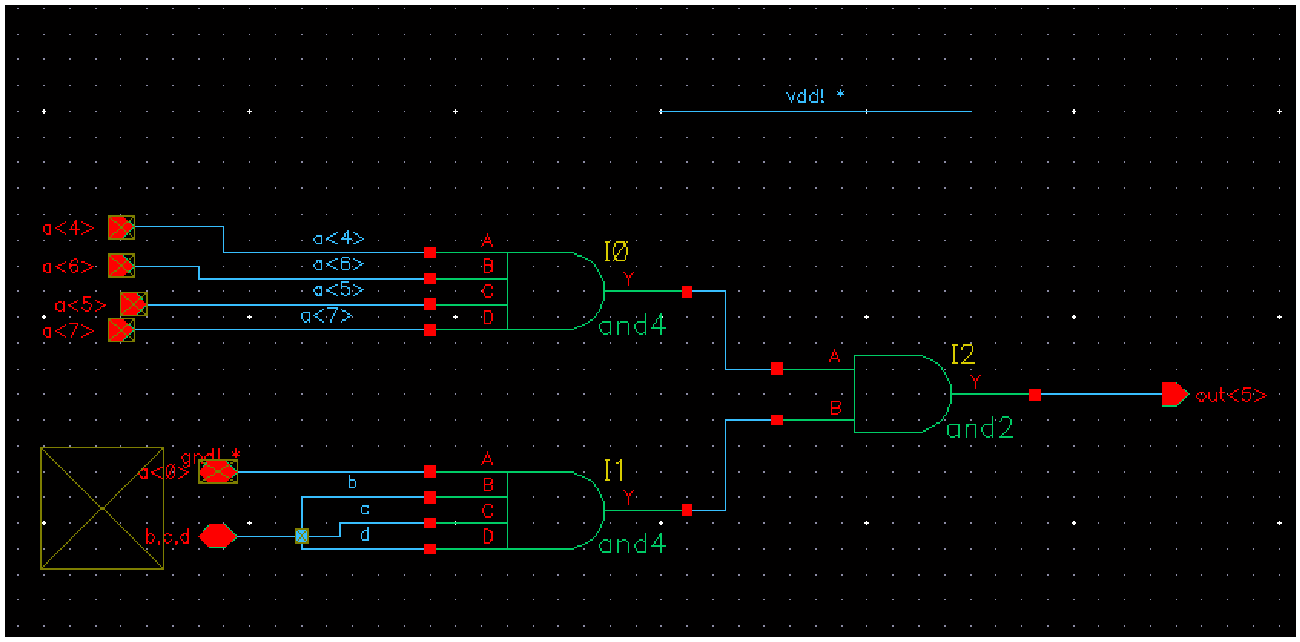
Consider the following example. The placed master of a cell, `mycell`, is as follows:



The terminals in the symbol are:

```
a[0], a[4:7], b, c, d, out[5]
```

The schematic of the cell, `mycell`, is as follows:



The terminals in the schematic are:

```
a[0], a[4], a[5], a[6], a[7], b, c, d, out[5]
```

Expand on Mismatch

If you choose the default option to expand on mismatch, the terminals are expanded wherever there is a mismatch in the name. In the given example, terminal a[4:7] of the instance of mycell has been expanded. The generated netlist is given below:

```
module mycell ( out[5], {b, c, d}, inh_mygnd, inh_myvdd, a[4], a[5], a[6], a[7] );
inout  inh_mygnd, b, c, d;
output [5:5]  out;
input [4:7]  a;
inout  inh_myvdd;
specify
    specparam CDS_LIBNAME  = "mylib";
    specparam CDS_CELLNAME = "mycell";
    specparam CDS_VIEWNAME = "schematic";
endspecify
and I1 ( net26, inh_mygnd, b, c, d);
and I0 ( net31, a[4], a[6], a[5], a[7]);
and I2 ( out[5], net31, net26);
endmodule

module topcell ( out1, in1[0], in1[5:7], {in2[0], in1[1:3]} );
output  out1;
input [0:0]  in2;
input [0:7]  in1;
specify
    specparam CDS_LIBNAME  = "mylib";
    specparam CDS_CELLNAME = "topcell";
    specparam CDS_VIEWNAME = "schematic";
endspecify
mycell I0 ( out1, in1[0], in1[5:7], in1[0], cds_globals.vdd_, in2[0], in1[1],
in1[2], in1[3]);
endmodule
```

You can use the Expand on Mismatch option with *Netlist Explicitly* option on the *Netlist Setup* form to generate explicit netlist, but the resulting netlist might not be a purely explicit. Therefore, if you want to generate a pure netlist, it is recommended to use the Merge All option instead.

Honor Switch Master

With implicit netlisting, you can choose to honor the switch master at both module and instance level to synchronize the terminals. In this case, the netlister merges/expands the ports of the placed master to match with the switch master. If some extra terminals are found on the placed master, the netlister ignores those terminals and generates netlist. However, if there are extra terminals on the switch master and no matching terminals are found on the placed master, the netlister gives errors.

For the above example, if the *Honor Switch Master* option is selected, the netlist will be generated as given below:

```
module mycell ( out, inh_mygnd, {b,c,d}, inh_myvdd, a[4], a[5], a[6], a[7] );
inout  inh_mygnd, inh_myvdd, d, c, b;
output [5:5]  out;
input [4:7]  a;
specify
    specparam CDS_LIBNAME  = "mylib";
    specparam CDS_CELLNAME = "mycell";
    specparam CDS_VIEWNAME = "schematic";
endspecify
and I1 ( net26, inh_mygnd, b, c, d);
and I0 ( net31, a[4], a[5], a[6], a[7]);
and I2 ( out[5], net31, net26);
endmodule

module topcell ( out1, in1[5:7], in1[0], {in2[0],in1[1:3]} );
output  out1;
input [0:0]  in2;
input [0:7]  in1;
specify
    specparam CDS_LIBNAME  = "mylib";
    specparam CDS_CELLNAME = "topcell";
    specparam CDS_VIEWNAME = "schematic";
endspecify
mycell I0 ( out1, in1[0], in1[5:7], cds_globals.vdd_, in1[2], {in2[0],in1[1]},
in1[3]);
endmodule
```

Merge All

In this case, the netlister merges all the vector terminals to create pure buses. Scalar terminals are kept as is and the bundled terminals are converted to scalar ports.

Following are the special considerations for merging:

- While merging the terminals of a single bus, if there are holes (that means, if connections for some terminals of a bus are missing), the netlister inserts dummy connections in the netlist. For example, in the schematic view of the cell, mycell, shown above, the connections are defined only for terminals `a[0]`, `a[4]`, `a[5]`, `a[6]`, and `a[7]` of the bus `a`. For other terminals, the connections are missing. With Merge All option, the netlister createS dummy connections for such terminals to create a pure bus. By default, these dummy nets are named as `dummy_net_cadence` and the terminals are of type `input`. It is recommended that you should not create your own nets with this name.
- If the vector terminals being merged are of different directions, the netlister still merges them and considers them as inout terminals.
- If the switch master of an instance is a text view, the netlister automatically honors the switch master for that instance and creates a valid Verilog netlist.

For the example given above, if the *Netlist Explicitly* option is `on`, the netlist will be generated as given below:

```
module mycell ( out,
.a({inh_mygnd,dummy_net_cadence,dummy_net_cadence,dummy_net_cadence,a[4],a[5],a[6
],a[7]}), b, c, d, inh_myvdd );
inout  b, c, d, inh_myvdd, inh_mygnd;
input  dummy_net_cadence;
output [5:5]  out;
input [4:7] a;
specify
    specparam CDS_LIBNAME  = "mylib";
    specparam CDS_CELLNAME = "mycell";
    specparam CDS_VIEWNAME = "schematic";
endspecify
and I1 ( net26, inh_mygnd, b, c, d);
and I0 ( net31, a[4], a[6], a[5], a[7]);
and I2 ( out[5], net31, net26);
endmodule


module topcell ( out1,
.in1({in1[0],in1[1],in1[2],in1[3],dummy_net_cadence,in1[5],in1[6],in1[7]}), in2
);
output  out1;
```

```
input  dummy_net_cadence;
input [0:0]  in2;
input [0:7] in1;
specify
    specparam CDS_LIBNAME  = "mylib";
    specparam CDS_CELLNAME = "topcell";
    specparam CDS_VIEWNAME = "schematic";
endspecify
mycell I0 ( .out(out1),
.a({in1[0],dummy_net_cadence,dummy_net_cadence,dummy_net_cadence,in2[0],in1[1],in
1[2],in1[3]}), .b(in1[5]),
     .c(in1[6]), .d(in1[7]), .inh_myvdd(cds_globals.vdd_));
endmodule
```

In the above example, `dummy_net_cadence` are the dummy connections inserted for holes.

Creating Pure Explicit Netlist

To generate a pure explicit netlist, you can configure the following settings in the *Netlist Setup* form:

- Select the *Explicit Netlist* option
- Set the *Terminal Syncup* option to `Merge All`

When these two options are set, the netlister generates pure netlist for all the designs irrespective of the configuration of terminals between placed master and switch master. All types of terminals, split buses, bundles, or, mismatched terminals are merged.

While generating pure netlists, the netlister ignores extra terminals found at the instance, but if extra terminals are found at the switch master, the netlister flags errors.

Note: To generate a pure explicit netlist with flattened buses, set the Terminal SyncUp option to 'Expand on Mismatch'. You cannot generate a netlist with flattened buses if the Terminal SyncUp option is set to 'Merge All' or 'Honor Switch Master'.

Including Verilog Text Views in Netlist

To include verilog text views in the netlist, set the *Terminal SyncUp* option on the *Netlist Setup* form to `Honor Switch Master`. In this case, the netlister honors only the terminal definitions existing in the text view and creates these terminals as is at the instance line.

While generating netlists with Verilog text views, the netlister ignores extra terminals found at the instance, but if extra terminals are found at the switch master, the netlister flags errors.

Adding Simulation Properties

You can add the following simulation properties to instances you have placed in your design and to the schematic views of their master cells.

- Verilog properties that translate to Verilog *parameter* statements for master cells
- Verilog properties that translate to Verilog *defparam* statements for instances
- Delay and drive strengths for instances
- Netlister action (*nlAction*) properties for instances and master cells
- Ancillary data files (*hnlVerilogCellAuxData*) for schematic cell views

Properties Added for Imported Modules

When you use Verilog In editor to import an HDL Verilog module, the system automatically creates the following properties:

- portOrder
- paramOrder
- model Name
- verilogPrintDelayAndStrength
- verilogFormatProc
- timescale

The Verilog netlister uses these properties to control the port sequence, model name, formatting functions and timescale.

When all of these properties exist, the netlister formats the instance ports implicitly based on the portOrder property definition. You can use the Edit Pin Order form to rearrange the pin order.

The following list defines each property and provides an example for each property.

portOrder

portOrder	Displays a list that specifies the port ordering (pin names) by pin sequence.

```
syntax:         ("pinName1" "pinName2" "pinName3"....)
example         ("out" "in1" "in2")
```

If the portOrder property is not set for the `hnlVerilogPrintPrimGate`, `hnlVerilogPrintBehavePortOrder`, and `hnlVerilogPrintBehaveModel` formatting functions, then the default portOrder is used. The default portOrder is the Output Pins, followed by the InputOutput Pins, followed by the Input Pins, in the alphanumeric order.

paramOrder

paramOrder	Displays a list that specifies the order of Verilog parameter names. This needs to be added when `hnlVerilogDonotPrintDefparam` is `t`.

```
syntax:         ("paramName1" "paramName2" "paramName3" ....)
example         ("RISE" "FALL" "VDD")
```

Note: All the parameter names that will be used must be specified in the paramOrder list otherwise the Netlister will format using the default way, printing the Verilog defparam keyword.

```
module myinv ( Y, A );
output  Y;
input   A;

     parameter RISE = "10ps",
          FALL = "20ps",
          VDD  =  5.0;
....
....
endmodule
```

If `hnlVerilogDonotPrintDefparam` is `t`, then the instance I0 will be formatted something like this in the resulting netlist.

```
module buffer(Y,A);
output Y;
input  A;
myinv I0("25ns","35ns") (Y,A);
.....
endmodule
```

The Netlister will print the defparam values corresponding to the parameter names listed in the `paramOrder` property.

modelName

modelName	Displays the model name as a string value.

```
syntax:         "verilogModuleName"
example            "and"
instID->modelName
```

The netlister looks at the `modelName` property on the instance if the switch master of the instance is a stopping view and the properties, `verilogFormatProc` and hnlVerilogFormatInst Property are not defined on the master.

An exception to this rule is when the format procedures `hnlVerilogPrintPrimGate` or `hnlVerilogPrintBehaveModel` are defined on the master then also the netlister looks for the modelName property on the instance.

If `hnlVerilogIgnoreModelNameProperty` is set to `t`, Verilog netlister will ignore the `modelName` property while generating the netlist.

verilogPrintDelayAndStrength

verilogPrintDelayAndStrength	Displays delays and strengths of a cellview as a string value.

```
syntax:    "t | nil"
example    verilogPrintDelayAndStrength=t
```

If the `verilogPrintDelayAndStrength` property for a cellview is set to `t`, the `hnlVerilogPrintLibraryModel` procedure prints the delay and strength values of the cellview.

timescale

timescale	Displays the timescale property of the Verilog module.

```
syntax:         "<time_unit> / <time_precision>"
   <time_unit> = <order of magnitude><unit of measurement>
   <time_precision> = <order of magnitude><unit of measurement>
example         "1ns / 1ps"
            "10us/100ns"
```

Important

There is no space between the integer and the unit of measurement.

verilogFormatProc

verilogFormatProc	Displays the model name and port ordering for a cell as set.

```
syntax:        "verilogFormatFuncName"
example        "hnlVerilogPrintPrimGate"
```

Note: Only the name of the formatting function should be set as a value of the `verilogFormatProc` property. The arguments should not be specified with the function name. The function uses values from the properties.

Important

The formatter always looks first for a Verilog property called *hnlVerilogFormatInst Property*. When the formatter finds the *hnlVerilogFormatInst* property along with any other formatting properties, the *hnlVerilogFormatInst* property takes precedence over the other properties.

You can specify the following formatting functions for this property:

- hnlVerilogPrintBehaveModel
- hnlVerilogPrintVhdlImport
- hnlVerilogPrintBehavePortOrder
- hnlVerilogPrintPrimGate
- hnlVerilogPrintLogGate
- hnlVerilogPrintLibraryModel
- hnlVerilogPrintBufif0Notif0
- hnlVerilogPrintBufif1Notif1
- hnlVerilogPrintBidiXfr
- hnlVerilogPrintNmosPmos
- hnlVerilogPrintCmos

hnlVerilogPrintBehaveModel

Displays the model name and port sequence defined with the modelName and portOrder properties, respectively, of the cellview instance. If the portOrder property is not defined, the default portOrder is displayed. This function does not print strengths and delays.

This function is used with a user-defined model.

Example: `verilogFormatProc = "hnlVerilogPrintBehaveModel"`

hnlVerilogPrintVhdlImport

Displays the model name defined with the modelName property of the cellview instance. If the modelName property is not defined, the default value is displayed. If the model name contains special characters, '.' or ':', they are replaced with an '_'.

This function is used to print connectivity information for the instances imported through VHDL import .

Example: `verilogFormatProc = "hnlVerilogPrintVhdlImport"`

hnlVerilogPrintBehavePortOrder

Displays the port sequence as specified with the portOrder property of the cellview instance. If the portOrder property is not defined, the default portOrder is used.

The function affects the port order of the cell instance.

Example: `verilogFormatProc = "hnlVerilogPrintBehavePortOrder"`

hnlVerilogPrintPrimGate

Displays the model name and optional port sequence defined with the modelName and portOrder properties, respectively, of the cellview instance. If the portOrder property is not defined, the default portOrder is displayed. This function also prints strengths and delays.

If the property *str* is set to `R` on the instance, then it prefixes `r` to the primitive name taken from the `modelName` property. For example, if the modelName is *tranif1* and `str` is set to `R`, then the primitive name displayed is *rtranif1*. This function is used with Verilog primitives.

Example: `verilogFormatProc = "hnlVerilogPrintPrimGate"`

hnlVerilogPrintLogGate

Displays the model names defined with the modelName property and netlists them implicitly. This function is used with the following gates: nand, nor, and; as well as with other single-output logical gates.

If the property *str* is set to `R` on the instance, the function prefixes `r` to the primitive name taken from the `modelName` property.

Example: `verilogFormatProc = "hnlVerilogPrintLogGate"`

hnlVerilogPrintLibraryModel

Displays the model name and optional port sequence defined with the *modelName* and *`portOrder`* properties, respectively. The function also prints strengths and delays of the cellview, if the *`verilogPrintDelayAndStrength`* property is set to `t`.

This function is used with library models.

Example: `verilogFormatProc = "hnlVerilogPrintLibraryModel"`

hnlVerilogPrintBufif0Notif0

```
hnlVerilogPrintBufif0Notif0( t_gateType )
```

Displays *`bufif0`* or *`notif0`*. This function is used with `bufif0` or `notif0` gates. The ports are displayed in the following order:

```
output, input, not enable
```

The values for these ports can be specified in any of the following ways:
output - `y`, input - `a`, not enable - `en_`
or
output - `Y`, input - `A`, not enable - `EN_`

Argument: *`t_gateType`*, the type of gate where *`t_gateType`* is one of the valid values, enclosed in quotes.

Valid Values: `"bufif0" "notif0"`

Value Returned: None

Example:

```
verilogFormatProc = "hnlVerilogPrintBufif0Notif0("bufif0")
```

The above example prints the following instance line:

`bufif0 x(y,a,en_)`

where, *`bufif0`* is the model name, *`x`* is the name of instance and *`y`*, *`a`*, and *`en_`* are the names of nets connected to the output, input, and not enable ports.

hnlVerilogPrintBufif1Notif1

`hnlVerilogPrintBufif1Notif1(` *`t_gateType`* `)`

Displays *`bufif1`* or *`notif1`*. This function is used with `bufif1` or `notif1` gates. The ports are displayed in the following order:
output, input, enable

The values for these ports can be specified in any of the following ways:
output - `y`, input - `a`, enable - `en`
or
output - `Y`, input - `A`, enable - `EN`

Argument: *`t_gateType`*, the type of gate where *`t_gateType`* is one of the valid values, enclosed in quotes.

Valid Values: `"bufif1" "notif1"`

Value Returned: None

Example

```
verilogFormatProc = "hnlVerilogPrintBufif1Notif1("bufif1")
```

The above example prints the following instance line:

`bufif1 x(y,a,en)`

where, *`bufif1`* is the model name, *x* is the name of instance and *`y`*, *`a`*, and *`en`* are the names of nets connected to the output, input, and enable ports.

hnlVerilogPrintBidiXfr

`hnlVerilogPrintBidiXfr(` *`t_gateType`* `)`

Displays *tranif1* and *tranif0*. This function is used with tranif1 and tranif0 gates. If the property *str* is set to *R* on the instance, this function displays *rtranif1* and *rtranif0* gates.
The ports are displayed in the following order:
drain, source, gate.

The values for these ports can be specified in any of the following ways:
drain - `d`, source - `s`, gate - `g`
or
drain - `D`, source - `S`, gate - `G`

Argument: *`t_gateType`*, the type of gate where *`t_gateType`* is one of the valid values, enclosed in quotes.

Valid Values: `"tranif1" "tranif0" "rtranif1" "rtranif0"`

Value Returned: None

Example:

```
verilogFormatProc = "hnlVerilogPrintBidiXfr("tranif1")
```

The above example prints the following instance line:

```
tranif1 x(d,s,g)
```

where, *`tranif1`* is the model name, *`x`* is the name of instance and *`d`*, *`s`*, and *`g`* are the names of nets connected to the drain, source, and gate ports.

hnlVerilogPrintNmosPmos

`hnlVerilogPrintNmosPmos(` *`t_gateType`* `)`

Displays *pmos* or *nmos*. This function is used with pmos and nmos gates. If the property *str* is set to *R* on the instance, this function displays *rpmos* or *rnmos*.
The ports are displayed in the following order:
drain, source, gate.

The values for these ports can be specified in any of the following ways:
drain - `d`, source - `s`, gate - `g`
or
drain - `D`, source - `S`, gate - `G`

Argument: *`t_gateType`*, the type of gate where *`t_gateType`* is one of the valid values, enclosed in quotes.

Valid Values: `"pmos" "rpmos" "nmos" "rnmos"`

Value Returned: None

Example:

```
verilogFormatProc = "hnlVerilogPrintNmosPmos("pmos")
```

The above example prints the following instance line:

```
pmos x(d,s,g)
```

where, *pmos* is the model name, *x* is the name of instance and *d*, *s*, and *g* are the names of nets connected to the drain, source, and gate ports.

hnlVerilogPrintCmos

```
hnlVerilogPrintCmos( t_gateType )
```

Displays *cmos*. If the property *str* is set to `R` on the instance, this function displays *rcmos.*
This function is used with cmos gates.
The ports are displayed in the following order:
drain, source, ngate, pgate.

The values for these ports can be specified in any of the following ways:
drain - `d`, source - `s`, ngate - `gn`, pgate - `pn`
or
drain - `D`, source - `S`, ngate - `GN`, pgate - `PN`

Argument: *t_gateType*, the type of gate where *t_gateType* is one of the valid values, enclosed in quotes.

Valid Values: `"cmos" "rcmos"`

Value Returned: None

Example:

```
verilogFormatProc = "hnlVerilogPrintCmos("cmos")
```

The above example prints the following instance line:

```
cmos x(d,s,gn, pn
```

whe
nam

Add

Use

1.

2.

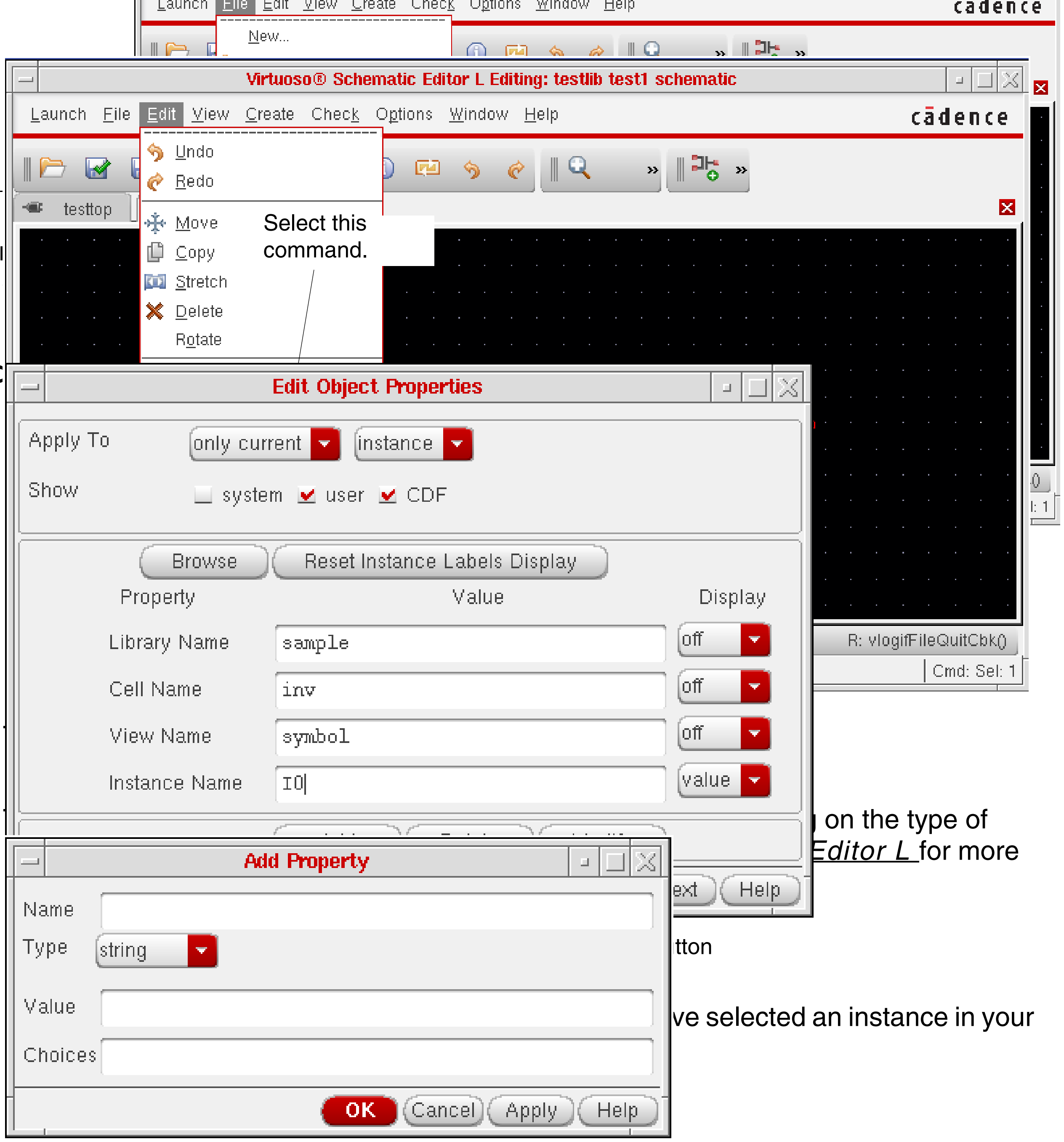
3.

on the type of
Editor L for more

4.

tton

ve selected an instance in your


Launch File Edit View Create Check Options Window Help
cadence
New...
Virtuoso® Schematic Editor L Editing: testlib test1 schematic
Launch File Edit View Create Check Options Window Help
cādence
Undo
Redo
Move
Copy
Stretch
Delete
Rotate
testtop
Select this command.
R: vlogifFileQuitCbk()
Cmd: Sel: 1
Help
ext

Edit Object Properties

Apply To	only current	instance
Show	☐ system	✓ user ✓ CDF

Browse | Reset Instance Labels Display

Property	Value	Display
Library Name	sample	off
Cell Name	inv	off
View Name	symbol	off
Instance Name	I0	value

Add Property

Name	
Type	string
Value	
Choices	

OK | Cancel | Apply | Help

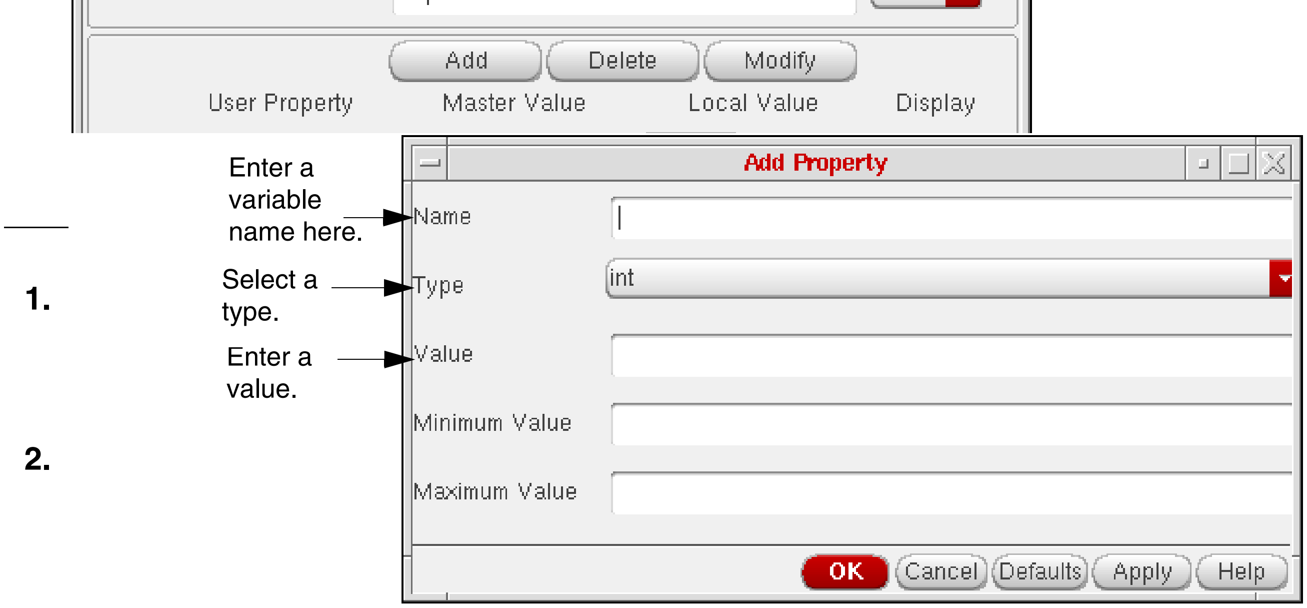
5. In the Name field, enter `verilog` in lowercase letters.
6. Select *hierProp* from the *Type* cyclic field.

   The Add Property form contracts.
7. Click *OK* on the Add Property form.

   The system updates the Edit Object Properties form.

Specifying Values for Verilog Properties

Use the following procedure to specify values for user-defined properties.



Name	
Type	int
Value	
Minimum Value	
Maximum Value	

1.

2.

3. Type a variable name in the *Name* field.

- ❑ When you are adding properties to master cells, enter the variable name that you want the system to write in a *parameter* statement.
- ❑ When you are adding properties to instances, enter the name of the variable whose value you want the system to write in a *defparam* statement.

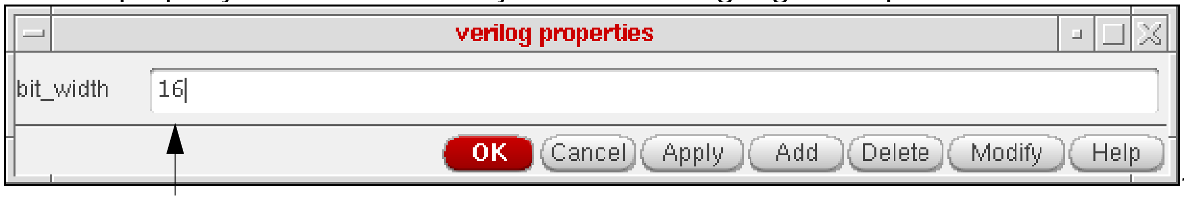
4. Select a type from the *Type* cyclic field.

5. In the *Value* field, enter the value you want to assign to the variable you named in the *Name* field.

Important

The system prints the value, without quotation marks, in the netlist if:

- ❑ the property name is not `COMPONENT`.
- ❑ the property value is a floating point number.
- ❑ the property value is a binary/octal/decimal string.
- ❑ the first and last character of the property value are '(' and ')' respectively.
- ❑ the property value matches any of the following regular expressions:



bit_width	16

6. nd

Repeat steps 1 through 6 to add other properties to the instance.

7. When you finish entering Verilog properties, click *OK* on the Verilog properties form.

When you renetlist your design, the system writes the instance properties as *defparam* statements.

Adding Delay Properties

Use the following procedure to add delay values to an instance of a leaf cell.

1. Make sure your design is editable.
2. In the Design Entry window, select the instance to which you want to add delay properties.
3. Select the *Edit – Properties – Objects* command.

   The Edit Object Properties form appears.

   The appearance of the Edit Object Properties form varies depending on the type of object you select in your design. Refer to the *Virtuoso Schematic Editor L* for more information about the Edit Object Properties form.
4. Click *Add* on the Edit Object Properties form.

   Note: The *Add* button appears on the form only if you have selected an instance in your design.

   The Add Property form appears.
5. In the *Name* field, enter the name of the delay property that you want to add to the instance.

   You can specify the delay properties listed in the following table. You must enter delay property names as lowercase characters.

Delay Property Name	What the Property Identifies
td	a single value for all delay types
tr	a rise delay
tf	a fall delay
tz	a turn-off delay

Note: The tr and tf properties are mutually exclusive to the td property. So, if you specify the tr and tf properties, the td property will be ignored. Otherwise, td gets printed.

6. Select *string* from the *Type* cyclic ield.
7. Enter the value for the delay property name specified.

Important

You must specify the delay value and its unit of measurement.

The following are Verilog units of measure.

Unit of Measure	What the Unit Identifies
s	seconds
ns	milliseconds
us	microseconds
ns	nanoseconds
ps	picoseconds
fs	femtoseconds

You can specify minimum, typical, and maximum delay values for tr, tf, and tz. The format is min:typ:max. You separate the values with a colon, as shown in the following example.

```
1ns:2ns:3ns
```

8. Click *OK* on the Add Property form.

   The system updates the Edit Object Properties form.

   If you want to enter multiple values, you can click *Apply* to submit the form and then modify the form for your next entry.

Note: The delay values you add to the instance in your design take precedence over delay values specified in the master cellviews.

Adding Drive Strength Properties

Use the following procedure to add drive strength properties to an instance of a leaf cell.

1. Make sure your design is editable.
2. In the Design Entry window, select the instance to which you want to add drive strength properties.
3. Select the *Edit – Properties – Objects* command.

   The Edit Object Properties form appears.

   The appearance of the Edit Object Properties form varies depending on the type of object you select in your design. Refer to the *Virtuoso Schematic Editor L* for more information about the Edit Object Properties form.
4. Click *Add* on the Edit Object Properties form.

   Note: The *Add* button appears on the form only if you have selected an instance in your design.

   The Add Property form appears.
5. In the *Name* field, enter the name of the drive strength you want to define for the instance.

   When you enter drive strengths, enter values for high and low strengths.

Drive Strength Name	Defines
Low_Strength	drive strength of logic 0
High_Strength	drive strength of logic 1

6. Select *string* from the *Type* cyclic field.
7. Enter the value of your first drive strength entry.
8. Click *OK* on the Add Property form.

   The system updates the Edit Object Properties form.

   If you want to enter multiple values, you can click *Apply* to submit the form and then either modify the form for your next entry or reselect *Apply*.

Note: The drive strengths you give to the instance in your design take precedence over drive strengths specified in the master cellviews.

Adding nlAction Properties

Netlister action (*nlAction*) properties control netlisting actions for individual cells.

Use the following procedure to add an *nlAction* property to an instance or to a master cell.

1. Make sure your design is editable.
2. In the Design Entry window, select the instance to which you want to add the property.
3. Select the *Edit – Properties – Objects* command.

   The Edit Object Properties form appears.

   The appearance of the Edit Object Properties form varies depending on the type of object you select in your design. Refer to the *Virtuoso Schematic Editor L* for more information about the Edit Object Properties form.
4. Click the *Add* button at the bottom of the Edit Object Properties form.

   Note: The *Add* button appears on the form only if you have selected an instance in your design.

   The Add Property form appears.
5. Type `nlAction` in the *Name* field.
6. Select *string* from the *Type* cyclic field.
7. In the *Value* field, enter the *nlAction* property you want to add to the instance.

   The following table summarizes *nlAction* properties you can add.

nlAction	Result	Apply to
stop	Netlists a cell as an instance definition. You must supply a module definition for the instance by specifying a Verilog library that contains the definition in the *Other Options* field of the Simulation Options form.	master cells
ignore	Prevents netlisting of individual instances.	master cells design instances

8. Click *OK* on the Add Property form.

The system updates the Edit Object Properties form.

If you want to enter multiple values, you can click *Apply* to submit the form and then modify the form for your next entry.

Adding Ancillary Data Files

You can add a property to a schematic cell view that forces the netlister to recognize ancillary data files as Verilog HDL input.

- This property is named *hnlVerilogCellAuxData* and takes a value of type ILList.
- The ancillary data files can include any Verilog command line or source code.

Ancillary data files can contain the *'include, 'ifdef*, *'else*, and *'endif* compiler directives and other constructs that are typical in files that are conditionally included in a design. You can control the inclusion of such files in a design by placing properties on schematics that force the netlister to read ancillary data files that contain the *+define* command line option or the *'define* compiler directive.

Use the following procedure to add ancillary data to a master view.

1. Make sure your design is editable.
2. Select the *Edit – Properties – Cellview* command.

   The Edit Cellview Properties form appears.
3. Click *Add*.

   The Add Property form appears.
4. Enter `hnlVerilogCellAuxData` in the *Name* field.
5. Select *ILList* from the *Type* cyclic field.
6. In the *Value* field, specify the ancillary data files using one of the following methods: absolute path names, relative path names, library cellviews.

   The system uses the files that you identify with absolute path names as the input for NC-Verilog, but the system does not copy these files into the run directory.

   The system also uses the files that you identify with relative path name or cell view information as the input for NC-Verilog. But the system copies these files into the `hdlFilesDir` directory in the run directory.

   To identify the ancillary data files that are cell views that resolve to text files, you provide the names for the library, cell, and view.

There are three formats for entering data files:

Filename Representation	**Type**	**Example**
Absolute path name	ILList	`(("/tmp/file.v"))`
Relative path name	ILList	`(("mux.v"))`
Library-Cell-View	ILList	`((("lib1" "inc" "functional")) nil)`

Observe the following rules when specifying path names and cell views:

- ❑ There is no limit to the number of files you can specify.
- ❑ Any combination of type formats is acceptable.
- ❑ The order of the types of items is irrelevant.
- ❑ Use spaces to separate multiple filenames.
- ❑ The quotation marks are essential parts of the syntax.

7. Click *OK* on the Add Property form.

The system updates the Cellview Properties form

Verify that you have indeed added the property by checking the CellView Properties form for correct list formatting. Be sure to review the CIW log after netlisting to ensure that there was no error in handling ancillary data files. A missing or incorrect property could result in a simulation that is incorrect.

Netlisting Switch-RC Cells

A Switch-RC property is an attribute that you can attach to a data object to store timing information for simulators.

Before you can netlist Switch-RC cells, you must add Switch-RC properties to your design and set the netlister to recognize the properties.

Setting the RC Data Variable

By default, the netlister ignores properties associated with Switch-RC switches and nets. To netlist the Switch-RC properties, you must set the `simVerilogHandleSwitchRCData` variable to `t`.

There are three ways to set the value for the variable.

- Select the Netlist SwitchRC option on the Netlist Setup form
- Edit the value in your `.simrc` file
- Specify the value in the CIW command line

Entering Switch-RC Properties

To enter Switch-RC properties, you use the *Add* option on the Edit Properties form. You access the Edit Object Properties form from the 77 menu of the schematic editor.

Refer to the *Virtuoso Schematic Editor L* for more information about using the Edit Object Properties form.

Formatting Functions for MOS Cells

You can use the following functions to format MOS cells:

- `hnlVerilogPrintNmosPmos`
- `hnlVerilogPrintCmos`
- `hnlVerilogPrintBidiXfr`
- `hnlVerilogPrintBufif0Notif0`
- `hnlVerilogPrintBufif1Notif1`

Switch-RC Instance Properties

You can add the following properties to instances in your design to specify Switch-RC attributes.

- algorithm
- technology

- width, length (w, l)
- driveStrength
- hnlVerilogCDFdefparamList

Refer to the *Virtuoso Schematic Editor L* for more information about adding properties.

algorithm

Use this property to specify the Switch-RC algorithm (*resistive*).

When you do not specify an algorithm, the netlister assumes the default (that is, *'default*).

Property:	algorithm
Type:	string
Valid Values:	"resistive" "NC" "default"

The netlister always looks for the instance first, followed by the switch master.

technology

Use this property to specify the technology.

When you specify the *resistive* algorithm, but do not specify a technology property, the netlister assumes the default (that is, *'switch resistive default*).

Property:	technology
Type:	string
Valid Values:	(user defined)

The netlister always looks for the instance first, followed by the switch master.

width, length (w, l)

Use these properties to specify the length and width of the switch. You must use the same units as those assigned to the *technology* property.

Property:	width, length *or* (w, l)
Type:	floating number or string
Valid Values:	(user defined)

The netlister always looks for the instance first, followed by the switch master.

driveStrength

Use this property to specify the strength of a mos cell.

Property:	driveStrength
Type:	fixed number
Valid Values:	1 < num > 250

hnlVerilogCDFdefparamList

Use this property on the instance, CDF, or the switched master of an instance to print CDF properties by using the *defparam* statement. The property when defined at the instance and CDF level should be of string data type but when defined on the switched master, it should be of SKILL list type. For example, you define it on the instance or CDF as,

```
hnlVerilogCDFdefparamList = Asim Lsim Wsim l w
```

On the switched master, you define it as,

```
hnlVerilogCDFdefparamList = ("Asim" "Lsim" "Wsim" "l" "w")
```

Verilog evaluates the NLP expressions of the form atPar, pPar, dotPar, iPar and [@abc...], [+abc...], [.abc...], [~abc...] for all properties in `hnlVerilogCDFdefparamList` and generates warning messages in case of problems in evaluation. However, due to a limitation of the hierarchical netlister, atPar is treated as pPar and dotPar is treated as iPar.

Note: The `CDS_Netlisting_Mode` environment variable should be set to `Analog` for component parameter evaluation to take CDF properties into account. When the variable is set to `Digital`, CDF properties are not taken into account though it results in better netlisting performance.

Verilog first searches for this property on the instance, then the CDF, and then the switched master.

Property:	hnlVerilogCDFdefparamList
Type:	■ string on instance and CDF level ■ SKILL list on the switched master level

Switch-RC Net Properties

You can add the following properties to nets in your design to specify Switch-RC attributes.

- netType
- chargeDecay
- tr
- tf
- chargeStrength
- c(capacitance)
- highThreshold
- lowThreshold
- hnlVerilogCDFdefparamList

Refer to the *Virtuoso Schematic Editor L* for more information about adding properties.

To declare that the design includes user-defined RC switch properties that must be honored in the netlist, set `simVerilogHandleSwitchRCData` to `t` from the Netlist Setup form or in `.simrc`.

netType

Property:	netType
Type:	string
Valid Values:	*Default values:* "trireg" "tri0" "tri1" "supply0" "supply1" "triand" "trior" "wand" "wor" "wire" *Custom values:* The net data type values specified in the variable `hnlVerilogWireNetTypeList`.
Output Format:	Net type support

chargeDecay

Property:	chargeDecay
Type:	string
Valid Values:	2.0
Output Format:	#(tr, tf, chargeDecay)

tr

Property:	chargeDecay,tr,tf
Type:	string
Valid Values;	3.5
Output Format:	#(tr, tf, chargeDecay)

tf

Property:	chargeDecay,tr,tf
Type:	string
Valid Values:	4.2
Output Format:	#(tr, tf, chargeDecay)

chargeStrength

Property:	chargeStrength
Type:	string
Valid Values:	"small" "medium" "large"
Output Format:	(small)

c (capacitance)

Property:	c (capacitance)
Type:	floating number
Valid Values:	(user defined)
Output Format:	(* const real capacitance = 4.5)

highThreshold

Property:	highThreshold
Type;	floating number
Valid Values:	4.3
Output Format:	(* const real highthresh = 4.3)

lowThreshold

Property:	lowThreshold
Type;	floating number
Valid Values:	3.4
Output Format:	(* const real lowthresh = 3.4)

Using CDF Properties

A UNIX environment variable, CDS_Netlisting_Mode, controls how component description format (CDF) properties are interpreted during netlisting. You can set this variable in the .cshrc file. The variable can be set as `Analog`, `Digital`, and `Compatibility`.

The syntax for this variable is:

```
setenv CDS_Netlisting_Mode "Analog"
                     "Digital"
                    "Compatibility"
```

The following table gives the property search order for all the three modes, `Analog`, `Digital`, and `Compatibility`. It also tabulates the expression evaluation format in all the three cases.

Setting	**Property Search Order**	**Expression Evaluation**	
		CDF	**NLP**
Analog	Device CDF only	Y	N
Digital	1. Instance 2. Master CellView	N	Y
Compatibility	1. Device CDF 2. Instance 3. Master CellView	Y	Y

By default, the environment variable CDS_Netlisting_Mode is set to `Digital`.

CDF properties can be used by either creating a Verilog hierprop property or by using *`hnlVerilogCDFdefparamList`* parameter.

Important

When using the *`hnlVerilogCDFdefparamList`* parameter, to get correct results from the hierarchical evaluation of parameters, ensure that you set the CDS_Netlisting_Mode as `Analog`.

If an instance has CDF parameters, the user does not need to create a Verilog hierprop property to ask the Verilog netlister to print CDF properties by defparam statement. But the user does need to create a *hnlVerilogCDFdefparamList* property on the switched master of the instance cell.

```
property name: hnlVerilogCDFdefparamList
property type: list type
property value(example): (“Asim” “Lsim” “l” “w” “Wsim”)
```

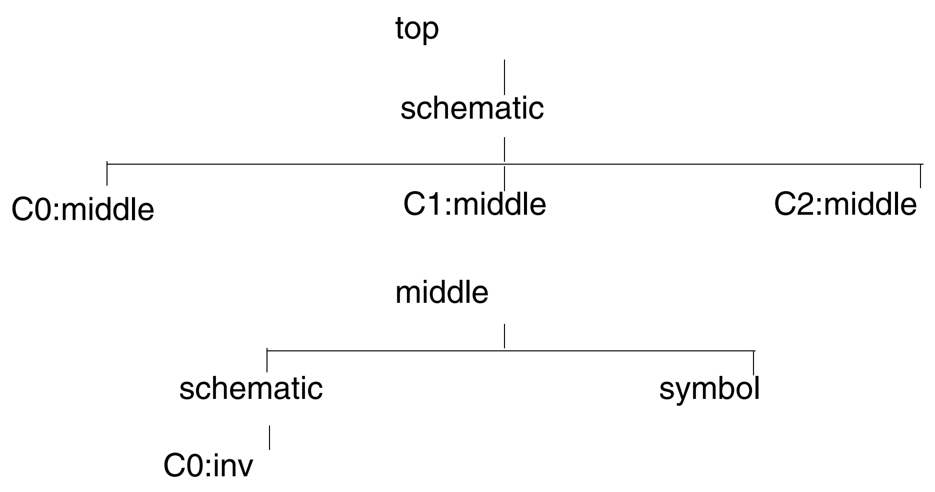
The Verilog formatter looks first at the instance Verilog hierprop properties and prints out those properties. Then, the formatter looks at the switched master of instance to determine whether property *hnlVerilogCDFdefparamList* exist.

If the property does exist, and if the master has a formatting property which instructs the netlister to print the instance as a logic gate, then CDF properties are not defined at the instance level using the `defparam` statement. However, this includes the fact that delay and strengths are always printed. Therefore, `tf td tr tz Low_Strength` and `High_Strength` can exist as CDF properties and get printed for logic gates.

If the switched master of instance is a non-stopping cell, then netlister prints the default CDF properties inside the `parameter` block of the Verilog module of this cell. The instantiated properties get printed as `defparam` statements in the module which instantiates this cell.

If a CDF property of an instance cannot be found by the netlister, the CDF property will be ignored.

For example consider the following hierarchy:



Cell `middle` has the following base CDF properties:

```
decayTime :

paramType : string
parseAsNumber : yes
units : "don’t use"
parseAsCEL : no
storeDefault : no
name : decayTime
prompt : decayTime
defValue : 10000

Cap Area:
```

```
paramType : string
parseAsNumber : yes
units : lengthMetric
parseAsCEL : yes
storeDefault : no
name : area
prompt : Cap Area
defValue : 100 M
```

```
Techno Unit :
```

```
paramType : string
parseAsNumber : yes
units : "don’t use"
parseAsCEL : yes
storeDefault : no
name : unit
prompt : Techno Unit
defValue : 1.25E-15
```

```
ratio:
```

```
paramType : float
storeDefault : no
name : ratio
prompt : ratio
defValue : 2
```

Cell `inv` has following `tr`, `tf` and `strength` CDF properties:

```
Low_Strength:
```

```
paramType : string
parseAsNumber : no
parseAsCEL : no
storeDefault : no
name : Low_Strength
prompt : L_St.
defValue : strong0
```

```
High_Strength:
```

```
paramType : string
parseAsNumber : no
parseAsCEL : no
storeDefault : no
name : Hign_Strength
prompt : H_St.
defValue : strong1
```

```
tf:
```

```
paramType : string
parseAsNumber : no
parseAsCEL : no
storeDefault : no
name : tf
prompt : tf
defValue : 0ns
```

```
tr:
```

```
paramType : string
parseAsNumber : no
parseAsCEL : no
storeDefault : no
```

```
name : tr
prompt : tr
defValue : 0ns
```

Also, cell `inv` has a Verilog view that has a `not` formatting instruction.

The following are how CDF properties are instantiated in cell top view schematic:

```
C0:

decayTime         -> 10000
Cap Area         -> 100 M
Techno Unit         -> 1.25E-15
ratio          -> 5
C1:

decayTime         -> 20000
Cap Area         -> 100 M
Techno Unit         -> 1.25E-15
ratio         -> 7
C2:

decayTime         -> 15000
Cap Area         -> 100 M
Techno Unit         -> 1.25E-15
ratio         -> 9
```

The following are the CDF properties on `inv` in `middle schematic`:

```
L_St.          -> strong0
H_St.         -> strong1
tf          -> 2ns
tr               -> 3ns
```

Now, since CDF properties need to be searched from both device CDF and instance CDF, you need to set the *`CDS_Netlisting_Mode`* env variable to `Compatibility`.

Netlisting cell `top` view schematic gives the following output:

```
// Library - insideLib, Cell - top, View - schematic
// LAST TIME SAVED: Jun 15 16:04:55 1999
// NETLIST TIME: Jun 15 16:23:09 1999
'timescale 1ns / 1ns

module top ( OUT1, OUT2, OUT3, in1, in2, in3 );
output  OUT1, OUT2, OUT3;
input  in1, in2, in3;

specify
   specparam CDS_LIBNAME  = "insideLib";
   specparam CDS_CELLNAME = "top";
   specparam CDS_VIEWNAME = "schematic";
endspecify

middle C2 ( OUT3, in3);

defparam
   C2.decayTime =  "15000",
   C2.area =  "100",
   C2.unit =  1.25e-15,
   C2.ratio =  9;
```

```
middle C1 ( OUT2, in2);

defparam
   C1.decayTime =  "20000",
   C1.area =  "100",
   C1.unit =  1.25e-15,
   C1.ratio =  7;

middle C0 ( OUT1, in1);

defparam
   C0.decayTime =  "10000",
   C0.area =  "100",
   C0.unit =  1.25e-15,
   C0.ratio =  5;
endmodule

// Library - insideLib, Cell - middle, View - schematic
// LAST TIME SAVED: Feb 15 17:27:34 1999
// NETLIST TIME: Jun 15 16:23:09 1999
‘timescale 1ns / 1ns

module middle ( OUT1, IN1 );
output  OUT1;
input  IN1;

parameter
   decayTime =  "10000",
   area =  100000000,
   unit =  1.25e-15,
   ratio =  2;

specify
   specparam CDS_LIBNAME  = "insideLib";
   specparam CDS_CELLNAME = "middle";
   specparam CDS_VIEWNAME = "schematic";
endspecify

not ( strong0,strong1 ) #(3, 2)  C0 ( OUT1, IN1);

endmodule
```

Note: Any CDF property whose unit is of type `lengthMetric` would be scaled by the hnl variable hnlVerilogLenWidDefaultScale. The default value of this variable is 1e-06. In the example above, this scale has been set to 1.

Printing CDF Parameters in Inline Explicit Format

If an instance has CDF parameters as well as Verilog parameters, both the parameters can be read during netlisting. For the CDF parameters to be printed in the inline explicit format, the following flag needs to be set to true in the `.simrc` file or at CIW:

```
hnlVerilogPrintCDFParamExplicit = t
```

The `hnlVerilogPrintCDFParamExplicit` flag, in turn, requires the following flags to be set to true:

```
hnlVerilogDonotPrintDefparam = t
simVerilogPrint2001Format = t
```

To print CDF parameters like Verilog parameters on the instance line, define the CDF parameter `hnlVerilogCDFdefparamList` with the following setting:

```
paramType: string
parseAsCEL: yes
name: hnlVerilogCDFdefparamList
prompt: hnlVerilogCDFdefparamList
defValue: <names of all/subset of already-defined CDF parameters separated by
space>
display: nil
```

Note: Verilog parameters and CDF parameters can have the same name.

When an instance with both CDF as well as Verilog parameters is encountered, the following rules are followed for netlisting:

- Verilog parameters are given precedence over CDF parameters. Additionally, the values of Verilog parameters override the values of CDF parameters.
- Verilog parameters follow the order of precedence defined in the *`paramOrder`* property.
- If the value of a CDF parameter is changed for an instance, then the parameter with the new value is netlisted.

Note: Switch Pcell parameters are always printed in inline explicit format irrespective of whether they are overridden at instance line with the `hnlVerilogPrintOverriddenCDFParamOnly` flag.

Note: For designs in which netlist is not possible in explicit format, `tmpFileForImplicitConnInExplicitForm` file is generated in the run directory and it consists of instance line information in explicit format.

Formatting Netlists

The Verilog netlister lets you format the netlisting output through the following properties.

- hnlVerilogFormatInst Property
- verilogFormatProc Property

hnlVerilogFormatInst Property

hnlVerilogFormatInst	Controls the model name of the instance master and the port sequence for most gates. Place this property on the master. The formatter always looks first for this Verilog property (`hnlVerilogFormatInst`). When the formatter finds this property along with any of the other formatting properties, the `hnlVerilogFormatInst` property takes precedence over the other properties.

Syntax: `verilogFormatFuncName`

Examples: `hnlVerilogPrintPrimGate`

Values of *hnlVerilogFormatInst:*

- `hnlVerilogPrintPrimGate("modelName optional-Port-Sequence")`
- `hnlVerilogPrintBehaveModel("modelName optional-Port-Sequence")`
- `hnlVerilogPrintBufif1Notif1("bufif1 | notif1")`
- `hnlVerilogPrintBufif0Notif("bufif0 | notif0")`
- `hnlVerilogPrintBidiXfr("(r)tranif1 | (r)tranif0")`
- `hnlVerilogPrintCmos("(r)cmos")`
- `hnlVerilogPrintNmosPmos("(r)pmos | (r)nmos")`
- `hnlVerilogPrintLibraryModel("modelname" "port_sequence")`

Note: To print strength and delays if the Verilog view or instance has a property, use `verilogPrintDelayAndStrength`. However, strengths and delays will not be printed for `hnlVerilogPrintBehaveModel`.

Following is an example of how you can use this property:

A cell `and2` is instantiated in a design. If you want the master cell name to be `and` primitive of Verilog instead of the cell name `and2`, use the *hnlVerilogFormatInst* formatting instruction in the following syntax.

```
hnlVerilogPrintPrimGate("and out in1 in2")
```

This usage manipulates the model name as `and`. The port sequence is also controlled as `out in1 in2`.

Place this formatting instruction property *hnlVerilogFormatInst* on any view that will be used during netlisting. That would be the view that the netlister chooses from the order defined in the *Netlist These Views* field of the Netlist Setup form.

Once the view is selected, you can add the formatting instruction property. Do not add the property from the library browser menu. Instead, use the property editor, once you have opened the design for editing.

For example, consider a cell `c1` with the following views:

```
"symbol" "functional" "schematic"
```

In the Netlist Setup form the view list is as follows:

```
"functional"  "schematic"  "symbol"
```

Based on the view list, the netlister chooses the `functional` view of `c1`. Therefore, to control the model name and port sequence, the formatting instruction property should be placed on the `functional` view of `c1`.

verilogFormatProc Property

Use this function to display the model name and port ordering for a cell.

For details, see verilogFormatProc.

Netlisting AutoLayout Views

To generate a Verilog netlist from an `autolayout` or `abstract` view

1. Open the view(s) to be netlisted
2. What you do next, depends on which of the following two cases applies:
   - ❑ When a placed master view (`symbol` or `abstract`) contains more terminals than the switch master

     In the placed master, set the `nlAction` property to `ignore` for all the terminals that are allowed to mismatch.
   - ❑ When both the placed master (`symbol` or `abstract`) and the switch master view contain some common terminals that you do not want to include in the Verilog netlist

In the switch master, add the `nlAction` property to `ignore` for the terminals that you do not want to include in the Verilog netlist.

Note: When the `nlAction` property is added on the terminals, it is also required to set the `simCheckTermMismatchAction` property in the `.simrc` file or CIW. For more details, refer to the Skipping Terminals section in the *Open Simulation System Reference*.

3. Set these NC-Verilog netlisting options
   - ❑ Add `autolayout` and `abstract` to the *Netlist These Views* field.
   - ❑ Add `abstract` to the *Stop Netlisting at Views* field.

Customizing Test Fixture Variables

The following sections describe how to change variables to customize your test fixture.

Modifying the Test Module Name

The variable `simVerilogTopLevelModuleName` stores the test module name and top-cell instantiated instance name as `test.top`. To define your own module test name and top-cell instantiated instance name, define this variable in the CIW.

Modifying the Test Fixture Template Name

By default, the test fixture template for the top-level cell is called `testfixture.template`. The variable that stores the name of the template file is `vlogifTestFixtureTemplateFile`.You can change the default template name by setting this variable in the CIW or in the `.simrc` file.

Saving the Template File

To preserve a test fixture template under incremental netlisting, set the *Generate Verilog Test Fixture Template* option on the extended Netlist Setup form to `nil`. When you set the *Generate Verilog Test Fixture Template* option to `nil`, the netlister does not generate a new test fixture template file. When you set the *Generate Verilog Test Fixture Template* option to `t`, a new test fixture template overwrites the existing one.

Whenever you switch to another stimulus file (input test pattern file) as the current test stimulus, be sure to set the *Generate Verilog Test Fixture Template* option to `t` in order to generate a correct test fixture template file.

Accessing the Map Table in the Stimulus File

The variable `simVerilogPrintStimulusNameMappingTable` controls the printing of the stimulus name mapping table (a map table of VSE names to Verilog names) to the current test stimulus file. By default, the variable is set to `nil`.

Adding Design Kits to the Test Fixture File

The variable `simVerilogInsertTestFixtureString` defines any design kits as Verilog statements in the test stimulus file.

For example, the following statement entered in the CIW identifies the use of the delay calculator design kit.

```
simVerilogInsertTestFixtureString = "$XYZ_delay_calc(test.top,22,5.0,best)
```

As a result, the stimulus file `testfixture.verilog` contains the following statements:

```
initial
   begin
      $XYZ_delay_calc(test.top,22,5.0,best);
   end
endmodule
```

Customizing the Netlisting Variables

The Verilog formatter uses the following variables for netlisting. Many of these variables are predefined. You can redefine them as needed either through the CIW or in the `.vlogifrc` or `.simrc` file.

The formatting variables can be categorized as:

- Variables for Netlist Setup Form Controls
- Other Variables used by Verilog Formatter

Variables for Netlist Setup Form Controls

The following variables are set through the Netlist Setup form:

Variable Name	Description
simReNetlistAll	This variable specifies the netlisting mode. By default, netlisting mode is set as `nil`. You can also reset this by setting the *Netlisting Mod*e field on the *Setup — Netlist Setup* form.
verilogSimViewList	This variable specifies that for each cell, the netlister performs a search of the specified views in the order in which they appear. You can reset this flag by resetting the *Netlist These Views* field on the *Setup — Netlist Setup* form.
simVerilogLaiLmsiNetlisting	Specifies if all Lai_verilog or Lmsi_verilog cellviews to be placed first in the netlist. By default, it is set to `t`*.* You can reset this flag through the *Netlist For LAI/LMSI Models* field on the *Setup — Netlist Setup* form.

Variable Name	Description
simVerilogTestFixtureTemplate	A flag to control which test fixture stimulus file should be generated. By default, it is set to `All`, which means all test fixture files, such as `testfixture.verilog`, `testfixture.veritime`, and `testfixture.verifault` will be generated. ■ If it is set to `Verilog`, only `testfixture.verilog` will be generated. ■ If it is set to `Veritime` only `testfixture.veritime` will be generated. ■ If it is set to `Verifault`, only `testfixture.verifault` will be generated. ■ If it is set to `None`, none of the test fixture files will be generated.
vtoolsUseUpperCaseFlag	A flag to generate netlist in uppercase. You can reset this flag through the *Netlist Uppercase* field on the *Setup – Netlist Setup* form.
hnlVerilogCreatePM	A flag to create the pin mapping files necessary to convert Standard Delay Files (SDF) pin names to NC-Verilog pin names. You can reset this flag through the *Generate Pin Map* field on the *Setup — Netlist Setup* form.

Variable Name	Description
simVerilogFlattenBuses	A flag to determine if all the buses should be flattened. The value is set through the Preserve Buses field on the Setup -> Netlist Setup form. By default, it is set to `nil`. If set to `t`, all buses are flattened. If you run the Verilog netlister in standalone mode, you need to define this flag in the file `si.env`. **Note:** To ensure that explicit netlisting is done in the case of split buses, you need to set `simVerilogFlattenBuses` to `true`. if you do not set this flag to true, you may get a warning message indicating that the netlisting is not done explicitly because of a split bus.
simVerilogHandleSwitchRCData	A flag that specifies that netlisting includes user-defined RC switch properties. By default, it is set to `nil`. You can reset this flag through the *Netlist SwitchRC* field on the *Setup — Netlist Setup* form.
simVerilogProcessNullPorts	A flag to determine whether to print the internal net name. By default, it is set to `nil` to print the net name assigned to the floating instance port. If set to `t`, net name is not printed during instance port formatting.
simVerilogHandleUseLib	A flag to determine whether `t` the netlister automatically adds the `'uselib` directive to the netlist when a design includes two similarly named cells from two different libraries. By default, it is set to `nil`. You can reset this flag through the *Netlist Uselib* field on the *Setup — Netlist Setup* form.

Variable Name	**Description**
simVerilogDropPortRange	A flag to determine if a module port is printed without the port range. By default, it is set to `t` so that under implicit netlisting, the module port is printed without the port range. If there are split buses across module ports, this flag needs to be set to `nil` in order to get a correct netlist. If there is a partial pin terminal tapped out of an internal net with the same net name, the netlister will automatically set the flag to `nil`. If the module is instantiated within a schematic cell, its module instance will not be formatted explicitly, even when both the *Explicit Netlisting* option is enabled and the `hnlVerilogNetlistNonStopCellExplicit` variable is set to `t`. Under explicit netlisting, use of module ports with ranges in explicit connections is not supported. In this case, the Verilog netlist formatter automatically sets this flag to `t`. During instance formatting in explicit netlisting mode, the netlister will check the instance module to see whether the module has any 1-bit width signals. If so, implicit instance formatting will be used against such instance. According to NC-Verilog, explicit formatting against split bus, module port with range and bundle is not allowed. More details about Split Bus are available at the end of this chapter.
simHnlDropUnusedInheritedPorts	A flag to completely remove all unused inherited connections from a netlist. It is a boolean variable with a default value as `nil`. You can set this environment variable in the `.simrc` file. Set the value of the variable as `t` to remove the unused inherited connections.

Variable Name	Description
simVerilogIncrementalNetlistConfigList	A flag to determine which cellviews are included in the Netlist Configuration list. By default, it is set to `nil` and all cellviews in the design are included in the Netlist Configuration list. If set to `t`, only the cellviews in the design that need to be renetlisted under incremental netlisting are included in the Netlist Configuration list.
simVerilogGenerateSingleNetlistFile	A flag to generate a single Verilog netlist containing multiple modules instead of one netlist per module. By default, this flag is set to `nil`. If set to `t`, the netlister generates a single Verilog netlist file in the current simulation run directory with the name `netlist`.
hnlVerilogNetlistStopCellImplicit	A flag that suppresses printing of net name during instance port formatting. By default, it is set to `nil`. You can reset this flag through the *Symbol Implicit* field on the *Setup — Netlist Setup* form.
vlogifUseAssignsForAlias	A flag that specifies that the netlister uses an assignment statement for patches between nets. By default, it is set to `nil` and the netlister applies the default *cds.alias* to patches between nets. You can reset this flag through the *Assign For Alias* field on the *Setup — Netlist Setup* form.
vlogifSkipTimingInfo	A flag that causes the netlister to ignore timing information assigned to instances in the design. By default, it is set to `nil`. You can reset this flag through the *Skip Timing Information* field on the *Setup — Netlist Setup* form.

Variable Name	Description
`vlogifDeclareGlobalNetLocal`	A flag that allows you to declare global signals locally. By default, it is set to `nil` *and* the netlister uses the default signals (Global Power Nets and Global Ground Nets). You can reset this flag through the *Declare Global Locally* field on the *Setup — Netlist Setup* form.
`simVerilogNetlistExplicit`	The value of the *Netlist Explicitly* field on the Setup — Netlist Setup form. By default, it is set to `nil`. If set to `t`, explicit netlisting is performed. If you run the Verilog netlister in standalone mode, you need to define this flag in the file `si.env`. Even when you use this option, instances of behavioral modules will still be connected implicitly. To get explicit connections for behavioral modules use the hnlVerilogNetlistBehavioralExplicit variable.
`simVerilogEnableEscapeNameMapping`	A flag that causes the netlister to include escaped names in the netlist. You can reset this flag through the *Support Escape Names* field on the *Setup — Netlist Setup* form.
`hnlVerilogTermSyncUp`	A variable to determine how to synchronize terminals between an instance and its switched master. The default value of this variable is `nil`. You can set this variable to either `honorSM` or `mergeAll` in the `.simrc/.vlogifrc` file or the CIW. Alternatively, you can also select an option from the *Terminal SyncUp* list in the *Setup — Netlist Setup* form. When this variable is set to `mergeAll` and both `simIgnoreTerm` and `hnlVerilogIgnoreTerm` are unbound, the terminals for which the `nlAction` property is set to `ignore` are ignored while generating Verilog netlist.

Variable Name	Description
verilogSimStopList	This variable specifies the last levels of hierarchy needed for this netlist. If the current view is in this list, the view is netlisted and the hierarchy stops expanding. You can also set this variable through the *Stop Netlisting at Views* field on the *Setup — Netlist Setup* form.
simVerilogPwrNetList	This variable specifies the global net names you want netlisted with the supply1 wire type. Supply1 wire types are driven to logic state 1. You can also set this variable through the *Global Power Nets* field on the *Setup — Netlist Setup* form.
simVerilogGndNetList	This variable specifies the global net names you want netlisted with the supply1 wire type. Supply1 wire types are driven to logic state 1. You can also set this variable through the *Global Ground Nets* field on the *Setup — Netlist Setup* form.
simVerilogOverWriteSchTimeScale	This variable specifies that the defined Global Time Scale variables overwrite any time values or units defined within a schematic. You can also set this variable through the *Global TimeScale Overwrite Schematic TimeScale* field on the *Setup — Netlist Setup* form.
simVerilogSimTimeValue	This variable specifies a value for the global simulating time. You can also set this variable through the *Global Sim Time* field on the *Setup— Netlist Setup* form.
simVerilogSimTimeUnit	This variable specifies the unit for the global simulating time. You can also set this variable through the *Global Sim Time* field on the *Setup — Netlist Setup* form.

Variable Name	Description
simVerilogSimPrecisionValue	This variable specifies a precision value for the global simulating time. You can also set this variable through the *Global Sim Precision* field on the *Setup — Netlist Setup* form.
simVerilogSimPrecisionUnit	This variable specifies the unit for the global precision value. You can also set this variable through the *Unit* field on the *Setup — Netlist Setup* form.
simPSHierPrefix	Specifies the hierarchy prefix used in the testbench, such as `test.top` or `test:top`.
simPSSimulationDir	Specifies the path of the simulation database directory.
simPSSimulationFile	Specifies the path of the simulation database file.
simPSVerilogRunDir	Specifies the path of the Verilog run directory.

Other Variables used by Verilog Formatter

Variable Name	Description
hnlMaxLineLength	Specifies the maximum number of characters that a line in the netlist file can contain. If the number of characters exceeds the specified limit, the line splits and a line continuation character is placed at the end of the line. The Verilog netlister sets the value of this variable to `4000`. To accommodate more than 4000 characters in a line, appropriately set *hnlMaxLineLength* in `.simrc`. **Note:** Set *hnlMaxLineLength* to `nil` when you set and use the *hnlSoftLineLength* variable.

Variable Name	Description
hnlMaxNameLength	This variable describes the maximum number of characters allowed in a name. The default value of this variable is `50`. You can reset the value in the `si.env` file.
hnlSoftLineLength	Specifies the maximum length of a line of output after which folding and continuation of the line needs to be considered. The Verilog netlister sets the value of this variable to `72`.

hnlUserStopCVList	List of user specified cellviews, which are treated as stop views while netlisting a design. You can specify this list in the `.simrc` file. Although instances of such a cellview appear in a netlist, the cellview module is not printed in the netlist.
	
	In the example below, all the cellviews in the `libN` library will be treated as stop views. However, in the `lib1` library, only the `cell1`, `cell2`, and `cell3` cellviews will be treated as stop views.
	
	`hnlUserStopCVList = list`
	`(`
	`    ;;; all cells from this library`
	`    "libN"`
	`;;; cell1, cell2 and cell3 from lib1`
	`    list("lib1" "cell1" "cell2" "cell3"s)`
	`)`
	
	Note: The list should have only one entry for each library, listing all the cellviews that need to be treated as stop views.

Variable Name	Description

hnlUserIgnoreCVList	List of user specified cellviews, which are ignored while netlisting a design. You can specify this list in the `.simrc` file.
	
	In the example below, all the cellviews in the `libN` library will be ignored when a design is netlisted. However, in the `lib1` library, only the `cell1`, `cell2`, and `cell3` cellviews will be ignored.
	
	`hnlUserIgnoreCVList = list`
	`(`
	`    ;;; all cells from this library`
	`    "libN"`
	`    ;;; cell1, cell2 and cell3 from lib1`
	`    list("lib1" "cell1" "cell2" "cell3")`
	`)`
	
	Note: Each library should only have a single entry listing all the cellviews that need to be ignored during netlisting.

Variable Name	Description
hnlUserStubCVList	

Specifies the list of stub cellviews and the option to print those cellviews with their interface details to resolve any inherited connections or power and ground net expressions. An OSS-based netlister treats stub cellviews as stop cellviews when netlisting a Verilog design. The netlister does not print the netlist of the instances under the stub cellviews.

Based on the arguments provided, the netlister traverses the design hierarchy below the stub cellviews and selectively prints the stub cellviews and their interface details.

Specify the stub cellviews using the format given below.

```
hnlUserStubCVList = list(("lib1" "cell1" "view1"
"PrintAndTraverse"|"OmitAndTraverse"|"PrintAndStopTraversal")("
lib2" "" ""
"PrintAndTraverse"|"OmitAndTraverse"|"PrintAndStopTraversal"))
```

The string values can be interpreted as given below:

- `PrintAndTraverse` or `t` prints the instance line of the stub and its module definition, but without the instances under the stub cellviews, and continues traversing to identify inherited connections.
- `OmitAndTraverse` or `nil` or no value specified omits the stub from the netlist but prints only the instance line for the stub, and continues traversing to identify inherited connections. When no value is specified, the list contains only three elements.
- `PrintAndStopTraversal` prints the stub and stops traversing to improve performance.

If an empty string is provided for the library, cell, or view name, the specified format value is applied to all possible values for that field.

Example:

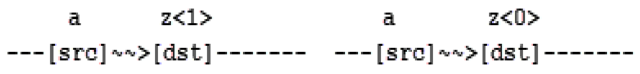
```
hnlUserStubCVList = '(("lib1" "" "view1" t) ("lib2" "cell2" "")
("" "cell3" "" "PrintAndStopTraversal") ("lib4" "cell4" ""
"OmitAndTraverse"))"
```

In this example:

- The entry `("lib1" "" "view1" t)` indicates that all the cells in the library `lib1` with the view name `view1` are treated as stub cellviews, which will be printed in the netlist with their interface details to resolve inherited connections.

Variable Name	Description
	■ The entry `("lib3" "cell2" "")` indicates that the specified format is applied to all possible views for `cell2` in library `lib3`. ■ The entry `("" "cell3" "" "PrintAndStopTraversal")` indicates that for all libraries and views within cells named `cell3`, the netlister prints the instance line of the stub and its module definition without instances, but stops traversing the design hierarchy below the stub cellviews. ■ The entry `("lib4" "cell4" "" "OmitAndTraverse")` indicates that the netlister omits all stub views in `cell4` of `lib4` and continues descending the design hierarchy to identify inherited connections. It prints only the instance line for the stub cellview. **Note:** If your design contains empty switch masters that are declared as stub cellviews, set `hnlEmptySwitchMasterAction` to `"honor"` to ensure that they are not ignored. For details on this HNL variable, see *Open Simulation System Reference*.
`hnlVerilogCellAuxData`	The relative path names, absolute path names, or library cellviews that identify ancillary data files as Verilog HDL input.
`hnlVerilogDeclareScalarSig`	Prints all the internal scalar signals as wires when the flag is set to `t`. Default: `nil`
`hnlVerilogDonotPrintDefParam`	A flag to avoid printing of defparams and control the formatting of the instance with the Verilog Hierprop. If this is set to `t` in CIW or the `.simrc` file, the netlister will print the defparam values using the inline parameter instantiation method. It is mandatory to add the “paramOrder property on either the instance or its master cellview containing the Verilog HierProp. Default: `nil`

Variable Name	Description
`hnlVerilogDumpIncludeFilesInNetlist`	When set to `t`, copies the content of an included HDL file directly to the netlist, instead of inserting an `` `include`` statement. When this variable is set to `t`, the content of the text cellviews in the design hierarchy are copied to the netlist. Any file specified in the *Pre-Module Include File* or *In-Module Include File* option of the SystemVerilog Integration Environment is also copied to the netlist. **Notes:** ■ This variable works only in single netlist file mode. ■ This variable does not support recursive inclusion of text files. Consider that `fileA.sv` includes `fileB.sv`. If you copy the contents of `fileA.sv` in the netlist using this variable, the contents of `fileB.sv` will not be copied in the netlist. Default: `nil`
`hnlVerilogGenErrForMixNetlist`	Displays an error message indicating a corrupt netlist in case of mixed netlists. A mixed netlist is one that contains a mix of explicit and implicit components. This happens when you have split busses but are netlisting explicitly. Since Verilog cannot explicitly netlist split busses, they are implicitly netlisted while the rest is explicitly netlist causing a mixed netlist. Set this flag to `t` if you wish to be warned in case a mixed netlist is being generated.
`hnlVerilogGetAuxFilesToPwd`	A flag to specify if auxiliary files need to be copied to the working directory. By default, it is set to `nil` and copies all auxiliary files to the directory *hdlDirFiles*.
`hnlVerilogGetMasterAlgorithm`	Whenever the formatter has to format a switch RC, it searches an instance for algorithm and technology by default. If you set this variable to `t`, then the formatter searches the switch master for ‘algorithm’ and ‘technology’. Default: `nil`

Variable Name	Description
hnlVerilogIgnoreTerm	A flag to specify that the terminals for which the `nlAction` property is set as `ignore` should be ignored while generating Verilog netlist. By default, it is set to `nil` and no terminal is ignored in Verilog netlisting.
hnlProcessAliasSignalWithSourceDirection	A flag that specifies that the netlister uses aliasing between more than two signals. By default, it is set to `nil`. It requires adding the `direction` property of the net, where the property value must be set to the source. The following example shows aliasing between the signals `a`, `z<0>`, and `z<1>`.  a z<1> ---[src]~~>[dst]------- a z<0> ---[src]~~>[dst]------- Here, if the source net is `a`, then the `direction` property must be set on net a, where the property value is set to source.
hnlVerilogIgnoreTermNameList	A flag to ignore certain cellview terminals during logical netlist generation, without the need to modify the relevant cellviews. To set this flag in `.simrc`, `si.env`, or Virtuoso CIW, specify the terminals to be ignored in the following syntax: `hnlVerilogIgnoreTermNameList=(list "`*`ignoreTermName1`*`" "`*`ignoreTermName2`*`" ...)` For details on how this variable is used for generating a logical Verilog netlist, see <u>“Generating a Logical Verilog Netlist by Setting Variables”</u> on page 204.
hnlVerilogIOInitStimulusStr	The initialized value of the in/out pin. By default, it is `z`. All in/out pins are set to `z` in the test fixture stimulus file.

Variable Name	Description
`hnlVerilogIOInitStimulusStr`	
	Use this variable to control the initial state of stimulus. For example, if you set `hnlVerilogIOInitStimulusStr = 0`, then the initial state of the inout stimulus is set to `0`. Default: `"z"`
`hnlVerilogMSUseAssign`	
	This flag is used only in mixed signal designs. If set to `t`, aliases are printed as assign statements, else they are printed as an instance of `cds_alias`. Default: `t`
`hnlVerilogNetlistBehavioralExplicit`	
	A flag to determine whether to netlist instances of behavioral modules explicitly. By default, it is set to `nil` and the netlister uses the implicit connections for behavioral modules. If set to `t`, the netlister uses explicit connection for behavioral modules. This works only if simVerilogNetlistExplicit has been set to `t`. If you use this flag, and the behavioral modules cannot be connected explicitly then you may get an incorrect netlist.
`hnlVerilogNetlistNonStopCellExplicit`	
	A flag to determine if the netlister uses the connection by name syntax for the non stopping cells. By default, it is set to `t` and the netlister uses explicit connection for both, stopping and nonstopping cells. If set to `nil`, the netlister uses the connection by name syntax for the stopping cells.
`hnlVerilogNetlistPrimGateExplicit`	
	The `hnlVerilogPrintPrimGate` formatting function always netlists implicitly. The flag `hnlVerilogNetlistPrimGateExplicit` when set to `t`, netlists explicitly, that is it prints the instance netlist statement using the Verilog connection by name scheme. Default: `nil`

Variable Name **Description**

hnlVerilogNetlistStopCellImplicit

Controls the printing of a stopped cell when netlisting in the implicit mode.

When set to `t`, it prints a stop cell implicitly. Otherwise, the stop cell is netlisted explicitly.

Default: `nil`

hnlVerilogPrimitiveList

List of standard primitives for which you cannot define the `hnlVerilogCDFdefparamList` property. The standard list is as follows:

```
hnlVerilogPrimitiveList = '( "tran" "tranif0" "tranif1" "rtran"
"rtranif0" "rtranif1" "pullup" "pulldown" "nmos" "pmos" "cmos"
"rpmos" "rcmos" "rnmos" "and" "nand" "nor" "or" "xor" "xnor" "buf"
"not")
```

You can edit the `hnlVerilogPrimitiveList` property to omit or include primitive names.

For example, if you want to use the `hnlVerilogCDFdefparamList` statement on the nmos primitive to print its properties in the netlist, make sure you delete the nmos primitive from the `hnlVerilogPrimitiveList` by redefining the `hnlVerilogPrimitiveList` variable.

hnlVerilogPrintSpecparam

Prints the specparams when set to `t` provided vlogif compatibility mode = '`4.2`'. When set to `nil`, specparams are not printed.

Default: `t`

Example:

```
specify
  specparam CDS_LIBNAME  = "verilog";
  specparam CDS_CELLNAME = "design";
  specparam CDS_VIEWNAME = "schematic";
endspecify
```

hnlVerilogPrintModulesByOrder

Controls the sorting of single netlist modules when they are printed in the netlist. When set to `t`, the modules are printed with alphabetical sorting.

Default: `nil`

Variable Name	Description
hnlVerilogRCSwitchUserDefineList	List of user-defined RC switch cells. If some switch cells are to be customized as one of the Verilog-defined switch types like *tranif1*, you can use this list variable to define those switch cells. For example: `hnlVerilogRCSwitchUserDefineList = '("nfet" "pfet")`
hnlVerilogRegroupLayoutSymbolBits	A switch that regroups instance terminal signals. By default, this switch is set to `off`. When set to `on`, through the `.simrc` file or CIW, the netlister regroups instance terminal signals according to the user defined port order formatting property of any Verilog functional cell.
hnlVerilogSkipTimescaleInNetlist	Controls the printing of the timescale directive in the netlist. When set to `t`, it skips the `timescale` directive. A warning note is printed in the netlist log file and in the CIW during netlisting to indicate that this flag is set and the timescale is skipped and that this may cause problems during simulation. Default: `nil` and timescale is always printed.
hnlVerilogSpecifyBlock	A flag to print `specify` blocks in a generated netlist. By default it is set to `t`. Setting this variable to `nil` will not print `specify` blocks in the netlist.
hnlVerilogSupportV93VhdlImport	A flag to support VHDL 93 version constructs in the VHDL foreign module that you want to co-simulate in Verilog integration. The default value of this flag is `nil`.

Variable Name	Description
hnlVerilogUseDefaultsForMapping	
	A flag when set to `t`, the values set by the users for the following mapping variables viz. `hnlMapNetFirstChar`, `hnlMapNetInName`, `hnlMapTermFirstChar`, `hnlMapTermInName`, `hnlMapBusFirstChar`, `hnlMapBusInName`, `hnlMapIfFirstChar`, `hnlMapIfInName`, `hnlMapInstFirstChar`, `hnlMapInstInName`, `hnlMapModelFirstChar` and `hnlMapModelInName` are ignored by the Verilog Netlister and the internal defaults are used for netlisting. The default value of this flag is `nil`. This can be useful when you want the Verilog netlister to ignore the following list set in the `.simrc` file. `hnlMapNetInName = list( '("<" "[") '(">" "]") '("!" ""))`
hnlVerilogUseFlattenTermName	
	If the names of the terminal and net connected to that terminal are different, and `simVerilogFlattenBus` and `hnlVerilogUseFlattenTermName` are set to `t`, the netlister uses the terminal names as port names in the cell definition.
hnlVerilogVerboseLevel	
	A flag to set the verbosity level of the netlister to display the type of messages. The variable can take three values: `error`, `warning`, or `info`. By default, the variable is set as `info` and all error, warning and info messages from netlister are displayed. **Note:** Messages generated by sources other than the Verilog formatter, such as the OSS traversal engine, cannot be controlled using the verbosity levels. They will be displayed regardless of the verbosity level set. Set this variable to `error` or `warning` to display only error or only error and warning messages, respectively.

Variable Name	**Description**

hnlVerilogWireNetTypeList

A flag to define the applicable netType values for wires in the following format.

```
hnlVerilogWireNetTypeList = list("dataType1" "dataTypeN")
```

Example:

```
hnlVerilogWireNetTypeList = list("trireg" "tri0" "tri1" "supply0"
"supply1" "triand" "trior" "wand" "wor" "wire" "reg")
```

Use Case Example:

Consider a design containing a wire `wire1` that is netlisted as shown below:

```
wire  [0:1]  wire1;
```

If you want to netlist this wire as data type `supply1`, instead of `wire`, do the following:

1. Set `simVerilogHandleSwitchRCData` to `t`.
2. Set `hnlVerilogWireNetTypeList` in `.simrc` as shown below.
   ```
   hnlVerilogWireNetTypeList = list("trireg" "tri0" "tri1"
   "supply0" "supply1" "triand" "trior" "wand" "wor" "wire"
   "reg")
   ```
3. Add the `netType` property with value `supply1` to the wire `wire1`. For details on adding properties, see *Virtuoso Schematic Editor L*.
4. Generate the netlist of the design using NC-Verilog Environment. The netlist has the following entry for `wire1`.
   ```
   supply1 [0:1]   wire1;
   ```

hnlVerilogUseHspiceDTermOrder

A flag to print the same split bus port order as the `hSpiceD termOrder` property. When this variable is set to `t` and the `termOrder` property exists in the `hSpiceD` CDF, the netlister uses the `termOrder` from `hSpiceD` CDF to print the term order.

simSupportDuplicatePorts

A flag to determine whether to remove duplicate ports from a netlist. By default, it is set to `t` and the simulator accepts duplicate ports. If set to `nil`, the netlister removes duplicate ports from a netlist.

Variable Name	Description
simVerilogBusJustificationStr	

A flag for selecting the bit order for buses. If this variable is not set or set to a value other than `R` or `L`, then the bit ordering defined for the bus is retained.

If the value is set to `R`, then the bit ordering is in descending order from MSB (most significant bit) to LSB (least significant bit).

```
/value to "R":
module testtop ( out1[3:0], out2[3:0], in[3:0] );

output [3:0] out1;
output [3:0] out2;
input [3:0] in;
...
test1 I2_1_ ( {X[6], X[7]}, in[3:2]);
test1 I2_0_ ( {X[4], X[5]}, in[1:0]);
test1 I0_1_ ( {X[0], X[1]}, in[3:2]);
test1 I0_0_ ( {X[2], X[3]}, in[1:0]);
```

In this example, any bus in the declaration, which is in the ascending order is modified to descending order and any bus in ascending order is split into individual bits in the connectivity part.

If the value is set to `L` then the bit ordering is in ascending order from LSB to MSB. Following is a sample netlist when you set the value to `L`:

```
/value to "L"
module testtop ( out1[0:3], out2[0:3], in[0:3] );
output [0:3] out1;
output [0:3] out2;
input [0:3] in;
...
test1 I2_1_ ( {X[6], X[7]}, {in[3], in[2]});
test1 I2_0_ ( {X[4], X[5]}, {in[1], in[0]});
test1 I0_1_ ( X[0:1], {in[3], in[2]});
test1 I0_0_ ( X[2:3], {in[1], in[0]});
```

In this example, any bus in the declaration which is in the descending order is modified to ascending order and any bus in descending order is split into individual bits in the connectivity part.

simVerilogIsConfigDesign

Set this flag to `t` if the top design is a config design. If set to `nil` then the netlister automatically checks for the top design.

Default: `nil`

Variable Name **Description**

`simVerilogNetlistFileCellList`

Prints all cell view names that have been netlisted in file.`simNetlistRunFile` in the run directory when set to `t`.

The format is as follows:

```
                                       simVerilogNetlistFileCellList
= '(( libName (<cellNameList> ) ) )
```

Example:

```
simVerilogNetlistFileCellList = '(("my_Lib"
     ("my_cell1" "my_cell2" "my_cell3")
  )
)
```

`simVerilogOverWriteSchTimeScale`

If `simVerilogOverWriteSchTimeScale` is set to `nil`, timescale is read from the cellview.

If set to `t` or timescale property is `nil` on the cellview, then it is calculated using the `simTimeUnit` and `simVerilogTimePrecisionVar` for all schematic viewed cells.

`simTimeUnit` is used to store the simulation time unit.

`simVerilogTimePrecisionVar` is used to store the simulation time precision. By default, they are set to a value of 1 nanosecond (timescale 1ns /1ns).

Default: 'timescale 1ns / 1ns'.

`simVerilogOverwriteVerimixStimulus`

Causes the Verilog netlister to generate a verimix stimulus file, `testfixture.verimix`. If set to `nil`, then the verimix stimulus file is created only if it does not exists at all. An existing file is not overwritten.

The default value is `nil`.

`simVerilogPrint2001Format`

A flag to control whether to print the netlist in the Verilog IEEE 1364-2001 format. To print the netlist in the Verilog IEEE 1364-2001 format, set this variable to `t`. By default this variable is set to `nil`.

Variable Name	Description
`simPrintInstAsBlackBox`	A flag to control whether to print the instance as a blackbox when the instance master library cell is listed in `hnlUserStopCVList`. When `simPrintInstAsBlackBox = t` and `hnlVerilogTermSyncUp` = `mergeAll` or `hnlVerilogTermSyncUp = honors`, the instance switch master is replaced by its place master. When `simPrintInstAsBlackBox = t` and `hnlVerilogTermSyncUp` is not set, the instance switch master is printed without any replacement. When `simPrintInstAsBlackBox = nil`, the instance switch master is printed.
`simVerilogPrintFormatInstTerm`	If a master cell has formatting instruction property, such as `hnlVerilogFormatInst`, the Verilog formatter tries to format each module port by printing the net that is connected to the port. If the port name is different from the net name to which it is connected, by default, the Verilog formatter declares the port explicitly, such as `.port(net_name)`. If this variable is set to `t`, then the formatter formats the module port by the port name instead of net name and does not use the explicit format. Default: `nil`
`simVerilogPrintStimulusNameMappingTable`	A flag to determine whether or not the stimulus name mapping table (mapping a VSE name to a Verilog name) should be printed to the test fixture Verilog stimulus file. By default, it is set to `nil` and the table is not printed. The Verilog stimulus file should be `testfixture.verilog`.
`simVerilogTopLevelModuleName`	Stores the top-level module name of the test. By default, it is set to `test.top`.
`simVerilogUseComposerPortOrder`	To print the default `portOrder` of VSE set this variable to `t` and set the cellview property `verilogFormatProc=hnlVerilogPrintBehaveModel`.

Variable Name	Description
vlogCdsAliasPortDirection	A variable to use the `cds_alias` module with a specific port direction. The values that you can specify for this variable are: `input`, `output`, and `inout`. For example, to specify an output port direction, set the variable in the `.simrc` file as follows: `vlogCdsAliasPortDirection = “output”` The default value of this variable is `input`.

vlogDropInstTermNetRanges	A variable to drop the bus range for complete nets connected to instance terminals.
	For example, if you have bus `net[0:6]` and complete bus is connected to terminal of an instance, then in the netlist for that instance, only bus name will be printed. That is, `net` instead of bus name with complete range, that is `net[0:6]`
	`ana2_busports_sv ana_inst2 (net);`
	However, if subset of bus is connected to instance terminal, then the bus name with range will be printed, that is, `net[2:6]`
	`ana3_busports_sv ana_inst3 (net[2:6]);`

vlogifCompatibilityMode	Determines the type of netlisting to be performed. By default, it is set at `4.2`, which is usually appropriate for Verilog simulation. You can set this flag to `4.0` if the generated netlist is to be used for Synergy, a logic synthesis tool.

Variable Name	Description
vlogifSkipTimingInfo	
	If `nil`, prints the delays for a basic logic gate in the Verilog netlist syntax. The full list of properties printed is: `[#( td )] or [#( tr,tf,tz )]` `td` is a single delay for both rise and fall delays. `tr`, `tf` and `tz` are the specific rise, fall, and high impedance delays if specified. If `td` is specified, it overrides the specification of `tr` and `tf`. The optional properties are enclosed in `[]`, and `or` means either of the properties may appear, but not both. Default: `nil`
vlogifTestFixtureTemplateFile	
	Stores the filename of the test fixture template. By default, it is set to `testfixture.template`.
vlogifUseAssignsForAlias	
	When set to `nil`, it creates a `cds_alias` module under the hdlFilesDir directory when a patch cord is used. When set to `t`, it prints all aliased signals using assign statements. Default: `nil`
vlogUseAssignForCdsThru	
	Converts `cds_thru` used to `assign` statements for shorted nets. Verilog In imports an assign statement as a `cds_thru` instance. The Verilog netlister prints it as a `cds_thru` instance line and not as an assign statement and this causes the exported netlist to differ from the imported netlist. To print assign statements instead of the `cds_thru` instance, set this flag to `t`. The `cds_thru` library cell information is specified using two SKILL variables, `vlogLibForCdsThru` and `vlogCellForCdsThru`. These variables are of string type and accept valid library and cell names as values. If these variables are not defined in the `.simrc` file or through CIW, instances of `basic/cds_thru` are replaced by default if the flag `vlogUseAssignForCdsThru` is set to `t`.

Variable Name	Description
vlogUseAssignForShorting	Handles the bi-directional propagation of the signals for shorted nets in the design by printing the `alias` assignment statement with the correct order of signal direction. The default value of this variable is `nil`, which means that the `assign` statement is printed for shorted nets. To print the `cds_alias` statements instead of `assign` statements, set `vlogifUseAliasForShorting` to `t`. The `cds_alias` statement manages the bidirectional flow, but this is not possible when the `assign` statement is used.

Note: In earlier releases, if the name of a terminal was different from the name of the net connected to that terminal because of an alias port, you were required to set the `simVerilogHandleAliasPort` flag to `t` to format the module port in an explicit connection. In addition, if the schematic had a bus port connected to an internal concatenated net with a different name, you also had to enable the *Explicit Netlisting* option. But now OSS implicitly handles an alias port connection. Therefore the `simVerilogHandleAliasPort` flag support has been removed.

File Structure Created by the Netlister

This section describes the directories and files that the system creates when you run the netlister in the NC-Verilog Integration Environment.

Directories

The system creates two directories in your current run directory:

- `hdlFilesDir`
- `ihnl`

The *hdlFilesDir* Directory

The `hdlFilesDir` directory contains ancillary data files. The `cds_global.v` and `cds_alias.v` modules, if they exist, are also located in this directory.

Module Name	**Description**
`cds_global.v`	If a net has a database property named *isGlobal*, it is considered a global net.You can specify any power and ground global signals in the *Global Power Nets* and *Global Ground Nets* fields on the Netlist Setup form. An HDL global module called `cds_globals.v` is created for these global nets. The global module itself defines the global power net using *supply1* and the global ground net using *supply0* declarations. The module defines all other global nets that have not been defined in either one of the global net fields by using wire declaration.

`cds_alias.v`	To handle a patch cord, the Verilog formatter generates an alias module `cds_alias.v`. The module is as follows:
	 `module cds_alias( cds_alias_sig, cds_alias_sig);`
	 `parameter width = 1;`
	 `input [width:1] cds_alias_sig;`
	
	 The syntax of an instance of this alias module is defined as follows:
	 `cds_alias [#`*`width`*`] cds_alias_inst(`*`signal`*`, `*`alias`*`);`
	 where
	 `width` = bus width
	 `signal` = primary signal name
	 `alias` = name of signal alias
	
	 Both *`signal`* and *`alias`* can be a combination of buses, concatenations, replications, and scalar signals.
	 `width` is optional when the width is 1 or when *`signal`* and *`alias`* are scalar nets.

The *ihnl* Directory

The `ihnl` directory contains “cell” directories. The cell directories correspond to the different cells in your design hierarchy. For example, if your design has four cells, then the cell directories under the `ihnl` directory are *cds0, cds1, cds2,* and *cds3.* Each cell has its own netlist definition and each cell directory contains three files: *netlist*, *map*, and *control*. For example:

```
ihnl/cellname0/netlist
          /map
          /control
```

Filename	Description
netlist	The description of the cell as a module in HDL
map	Map file that contains name-mapping data generated for this particular cell
globalmap	Map file for a cell’s global signals
control	File that determines whether or not IHNL should renetlist the related cells. This file should not be confused with the *control* file that exists in the simulation run directory. The *control* file in the simulation run directory is used during simulation and has no use during netlisting.

Files

When you run the netlister in a NC-Verilog Integration Environment, the system creates the following files in your run directory.

Filename	Description
control	Internal file created and used by NC-Verilog Integration
map	Map file for the netlist
raw	Internal file generated by OSS
si.env	Describes the environment where the last netlisting run was performed. This file is written in your run directory upon completion of a netlisting run. You can also use this file to run NC-Verilog netlisting as a standalone without invoking any Verilog user interface.
si.log	Log of operations performed when you netlist in background
testfixture.template	Top-level test fixture with stimulus template

Filename	Description
`testfixture.verifault`	Test stimulus file for Verifault
`testfixture.verilog`	Test stimulus file for NC-Verilog
`testfixture.veritime`	Test stimulus file for Veritime
`verilog.inpfiles`	List of Verilog input files from `ihnl` and `hdlFilesDir` directories
`config_file`	File to store cell binding and instance binding information when the NC-Verilog netlister uses an external Verilog file for binding a cell or an instance to a view in that referenced Verilog file. For details, see “Using a Verilog File Reference” on page 109 and *Referencing a Verilog File* in the *Virtuoso Hierarchy Editor User Guide*.

Split Bus

Case 1: Split bus across module ports

In this case, more than one port of a module share a common bus.

Verilog module is given below where the input is split into two parts `A[0:1]` and `A[2:4]`.

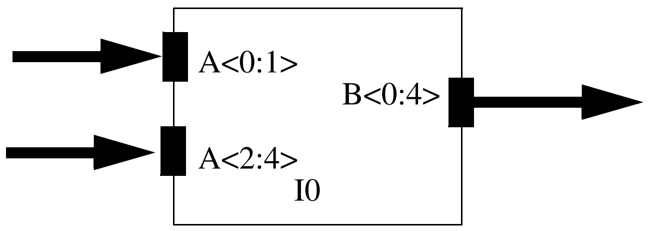
```
module splitbus( A[0:1],B[0:4],A[2:4] );
input A[0:4];
output B[0:4];

not I_1_( A[0], B[0]);
not I_2_( A[1], B[1]);
not I_3_( A[2], B[2]);
not I_4_( A[3], B[3]);
not I_5_( A[4], B[4]);

endmodule
```

Whenever, there is such an instance of a module whose terminals have split bus or bundles and the flag to netlist in explicit mode (connection by port name) is set, verilog netlister formats such an instance in implicit mode because the verilog language syntax doesn't support split bus or bundles for explicit port name connection.

In this case, *simVerilogDropPortRange* should be turned off in order to get a correct netlist.

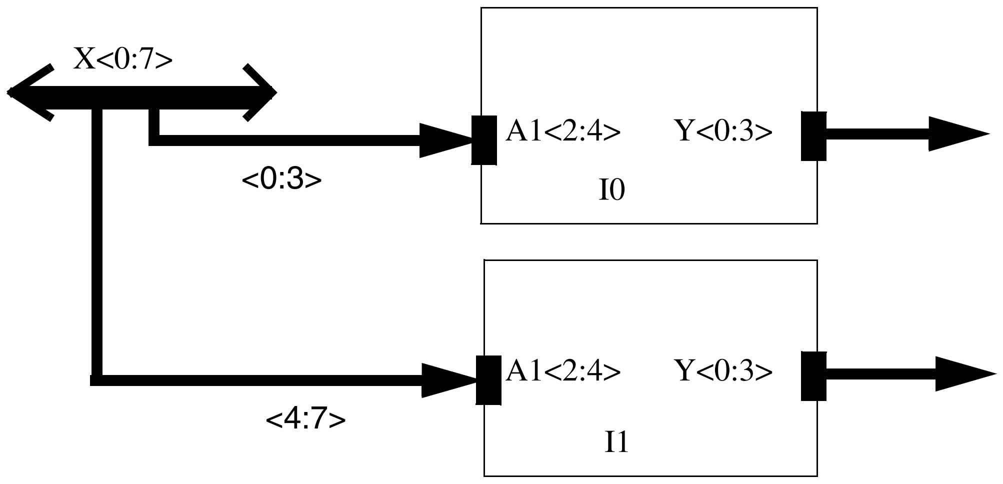
The corresponding schematic is shown below.

I0 Instance of `splitbus` module defined above.



Case 2: Bus Tap

In this case, bus X is shared between instances `I0` and `I1` where the *Most Significant Four Bits* of bus `X` is connected to terminal `A1` of instance `I0` and *Least Significant Four Bits* of bus `X` is connected to terminal `A1` of instance `I1`.



In this case, the netlisting will be explicit if generated in an explicit mode. The example given above shows that in this case if the explicit netlisting is turned on, the netlister will generate the explicit netlist.

As the syntax for explicit netlisting is `.TermName(NetName)`. This `termName` should be a unique identifier and only one identifier can be specified. In case there is a split bus (that is, multiple master terminals with same identifier base name and different range specifications), the `TermName` will not be unique or if the master terminal is a bundle, the explicit netlist will not be generated (as LRM does not allow this). Therefore, in this case implicit netlisting will be done.

Support for Verilog IEEE 1364-2001

From the IC5141 USR3 release, support for the following Verilog IEEE 1364-2001 constructs has been provided in Verilog Netlister. For more information, see *Verilog 2001 Support* chapter in the*Verilog® In for Virtuoso® Design Environment User Guide and Reference* manual.

- **Signed arithmetic**
  If a net or port has the property `verilogSignedDataType` set to `t`, then Verilog Netlister prints signed data types in the netlist for that net or port.
- **Sized and typed parameters and local parameters**
  Verilog Netlister prints explicit size and type declarations for parameters. It also prints local parameters.
- **Attributes**
  Attributes are supported for cellview, instances, ports, and nets.
- **Power Operator and Arithmetic Shift Operator**
  Verilog Netlister prints the power operator for exponential arithmetic as well as the arithmetic shift operators, `<<< and >>>` that are supported in the Verilog 2001 format.

Since the digital netlist generated by Verilog Netlister is also used by VERIMIX and the digital simulation in the VERIMIX flow is done by a simulator that is based on standard NC-Verilog simulator, Verilog IEEE 1364-2001 standard is not supported in the VERIMIX flow. Therefore, if you netlist in the Verilog IEEE 1364-2001 format and later use NC-Verilog for simulation, you get compilation error from the simulator.

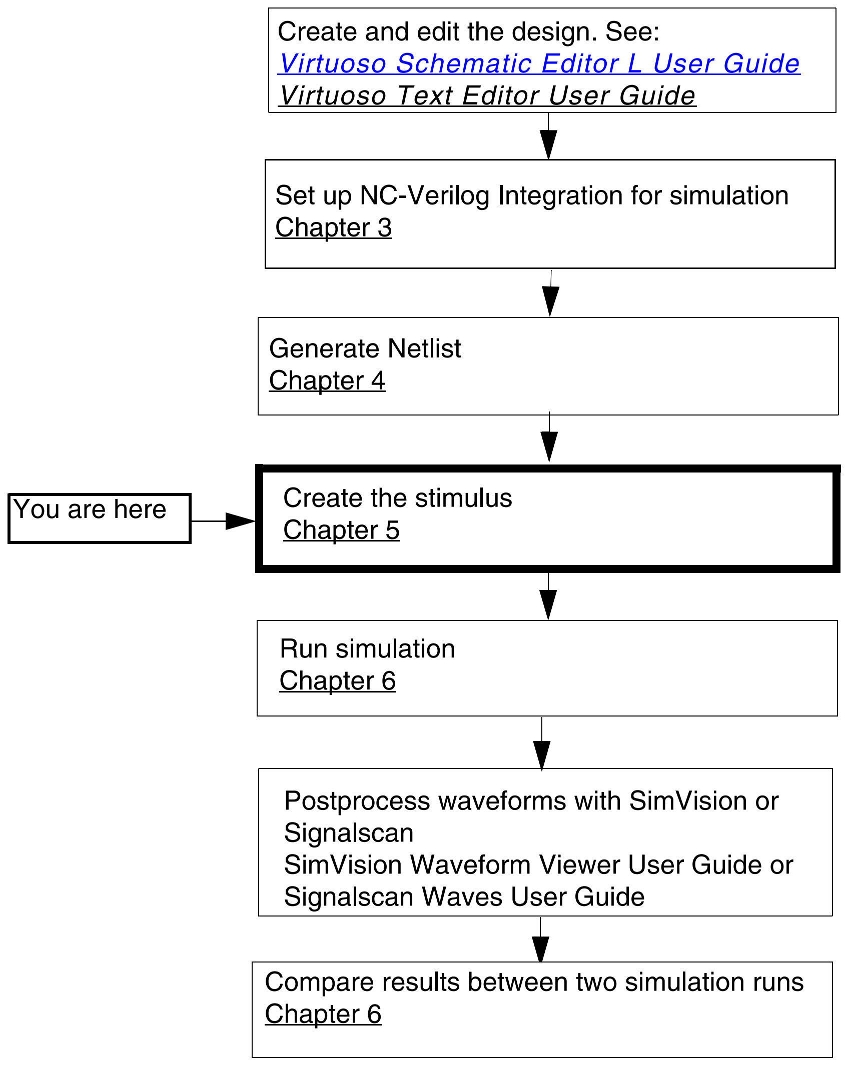
You can use the SKILL flag `simVerilogPrint2001Format` to decide whether to print in the Verilog IEEE 1364-2001 format. To enable Verilog netlister to print Verilog constructs in Verilog IEEE 1364-2001 format, set the SKILL flag `simVerilogPrint2001Format` to `t`.

6

Working with the Stimulus

In the NC-Verilog® Integration environment, a test fixture provides the stimulus that drives the simulation. This chapter describes the following:

Simulation Process Flowchart


Create and edit the design. See:
Virtuoso Schematic Editor L User Guide
Virtuoso Text Editor User Guide
Set up NC-Verilog Integration for simulation
Chapter 3
Generate Netlist
Chapter 4
You are here
Create the stimulus
Chapter 5
Run simulation
Chapter 6
Postprocess waveforms with SimVision or Signalscan
SimVision Waveform Viewer User Guide or Signalscan Waves User Guide
Compare results between two simulation runs
Chapter 6

Specifying the Test Fixture

To run a simulation, your current run directory must contain a test fixture. A test fixture provides stimulus to drive the simulation. In the NC-Verilog Integration Environment, a test fixture includes:

- A testbench file that contains instantiation of the complete design

- A stimulus file that contains preinitialized signal settings that drive a simulation

NC-Verilog Integration Environment saves the paths to these files in the following two variables in the `.vlogifrc` file:

- `vlogifCurrentTestFixture`: Saves the path to the current test fixture file.
- `vlogifCurrentStimulus`: Saves the path to the current stimulus file.

For information on selecting test fixtures, refer to Selecting Test Fixture Files on page 189. You can use either of the following methods to specify a test fixture:

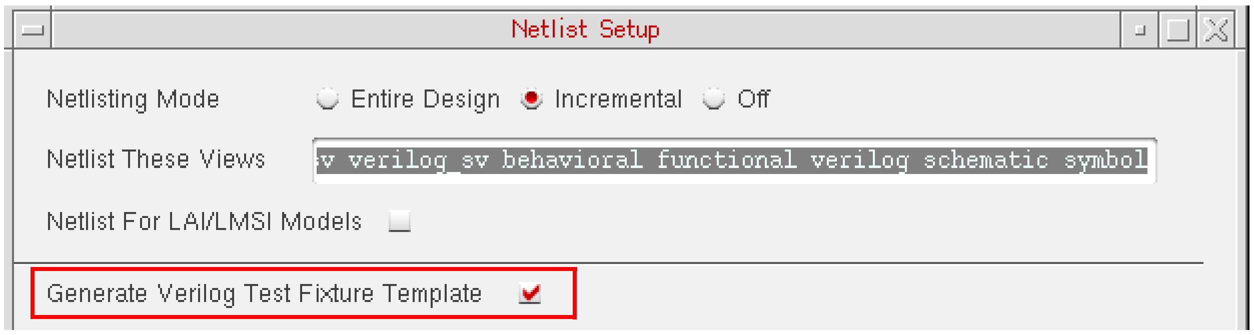
- Use Default Test Fixture Files
- Edit Existing Test Fixture Files

Using Default Test Fixture Files

By default, when you generate netlist for your design for the first time in a new run directory, NC-Verilog creates the following test fixture files:

- The default stimulus file, `testfixture.verilog`.
- The default testbench file, `testfixture.template`.

The test fixture files are automatically created when netlist is generated because the *Generate Verilog Test Fixture Template* check box on the *Netlist Setup* form is selected by default.



Netlisting Mode ○ Entire Design [✓ Incremental] ○ Off

Netlist These Views: v verilog_sv behavioral functional verilog schematic symbol

Netlist For LAI/LMSI Models ☐

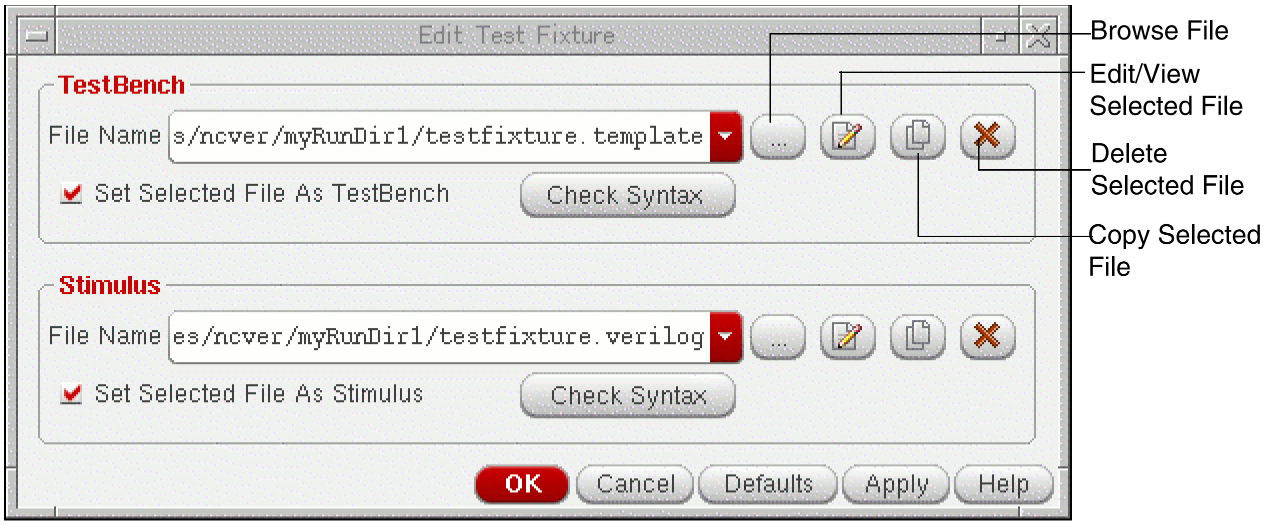
Generate Verilog Test Fixture Template ✓

Note: To use a test fixture that was used for the previous simulation of your design, deselect the *Generate Verilog Test Fixture Template* check box on the *Netlist Setup* form. In this case, the default test fixture files are not created while generation of netlist and the testbench and stimulus files saved in the directories specified in the `.vlogifrc` file are used.

Editing an Existing Test Fixture

If you do not want to use the default test fixture files created during netlist generation, you can select an existing set of files. If required, you can also modify the default or other selected text fixture files. In the NC-Verilog Integration Environment, you can use the *Edit Test Fixture* form to work with the test fixture files.

To open the *Edit Test Fixture* form, choose *Commands – Edit Test Fixture*.



File Name s/ncver/myRunDir1/testfixture.template
✓ Set Selected File As TestBench

File Name es/ncver/myRunDir1/testfixture.verilog
✓ Set Selected File As Stimulus

For more information on editing test fixtures, refer to Viewing or Editing Test Fixture Files on page 190.

Working with Test Fixture Files

The following sections describe how you can work with the test fixture files:

- Selecting Test Fixture Files
- Resetting the Default Test Fixture Files
- Viewing or Editing Test Fixture Files
- Copying Edit Test Fixture Files
- Deleting Test Fixture Files

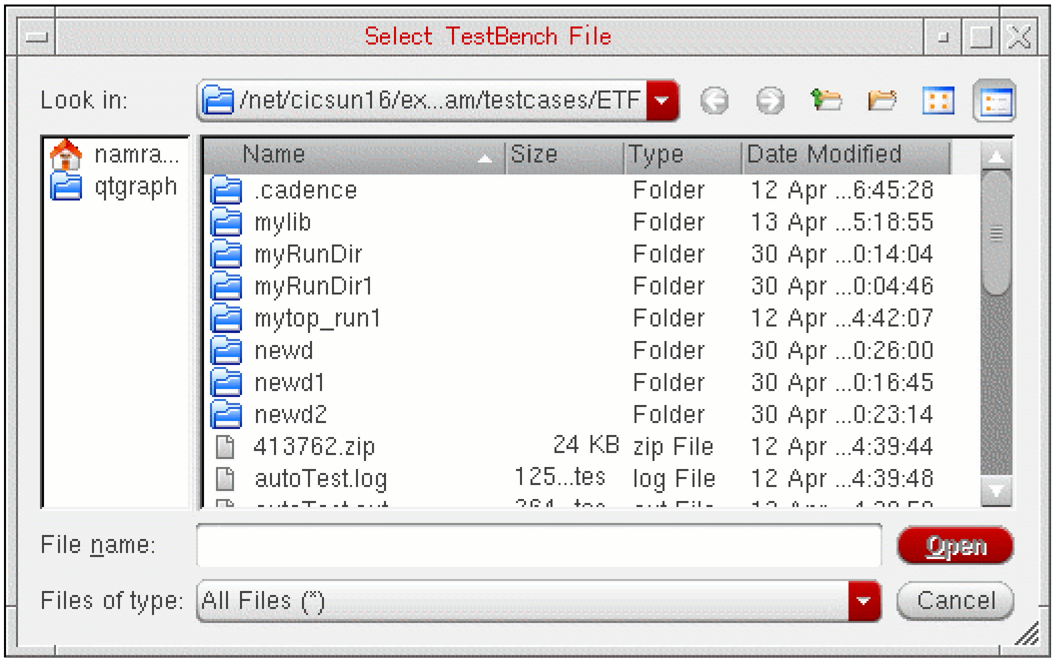
Selecting Test Fixture Files

The *File Name* field in the *Testbench* and *Stimulus* sections of the *Edit Test Fixture* form show the names of the testbench and stimulus files to be used for simulation. If you have generated netlist for your design for the first time in a new run directory, names of the default files are displayed. Otherwise, names of the files used in the last simulation run are displayed.

To select another testbench file:

1. In the *Testbench* group box, click *Browse File* ().

   The *Select Testbench File* form appears.



Select TestBench File

Look in: /net/cicsun16/ex...am/testcases/ETF

namra...
qtgraph

Name	Size	Type	Date Modified
.cadence		Folder	12 Apr ...6:45:28
mylib		Folder	13 Apr ...5:18:55
myRunDir		Folder	30 Apr ...0:14:04
myRunDir1		Folder	30 Apr ...0:04:46
mytop_run1		Folder	12 Apr ...4:42:07
newd		Folder	30 Apr ...0:26:00
newd1		Folder	30 Apr ...0:16:45
newd2		Folder	30 Apr ...0:23:14
413762.zip	24 KB	zip File	12 Apr ...4:39:44
autoTest.log	125...tes	log File	12 Apr ...4:39:48

File name:
Files of type: All Files (*)

2. Browse to the directory where the testbench you want to use is saved and select the file.

3. Click *Open* to use the specified file and close the form.

   The selected file becomes the current testbench and its name appears in the *File Name* field.

   Note: The names of the testbench files used in previous simulations are saved in the *File Name* field list. You can select a file name from this list also and use it for the current simulation.

4. Ensure that the *Set Selected File As Testbench* check box is selected. If this check box is deselected, the testbench file used in the previous simulation run is used.

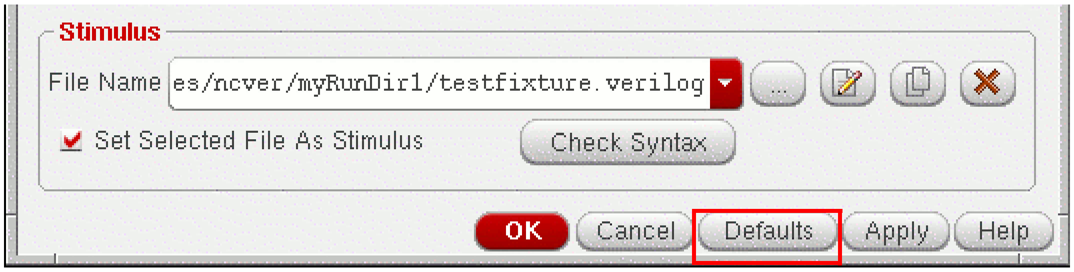
5. Click *Apply* to apply the change.

Note: The value of the `vlogifCurrentTestFixture` variable in the `.vlogifrc` file is also updated with the file name you selected in this form.

Similarly, in the *Stimulus* group box, you can select a stimulus file.

Resetting the Default Test Fixture Files

At any time, if you want to select the default test fixture files, click the *Defaults* button in the *Edit Test Fixture* form.



File Name: es/ncver/myRunDir1/testfixture.verilog
✓ Set Selected File As Stimulus

The names of the default files, `textfixture.template` and `textfixture.verilog`, are displayed in both the *File Name* fields in the form. In addition, the two check boxes, *Set Selected File as Testbench* and *Set Selected File as Stimulus*, are also automatically selected.

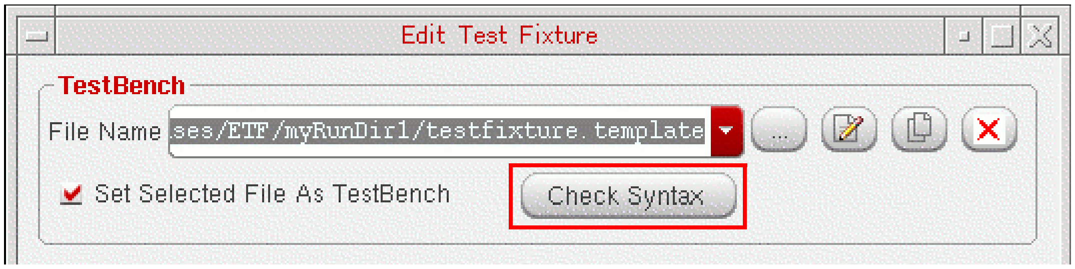
Note: If the default test fixture files were not created in any previous run, these are created when the next time netlist is created for this design.

Viewing or Editing Test Fixture Files

You can view or edit the contents of the currently selected test fixture files in a text editor. To open the testbench or stimulus file from the *Edit Test Fixture* form, click *Edit/View Selected File*( ).

The selected file opens in the default text editor of Virtuoso. You can make the desired changes in the file and save it. However, before making changes to a file, you might want to save a copy of that file. For more details, see Copying Edit Test Fixture Files.

After you edit a file, click *Check Syntax* below the file name to check for any syntax errors.


Edit Test Fixture
TestBench
File Name
ses/ETF/myRunDir1/testfixture.template
Set Selected File As TestBench
Check Syntax

NC-Verilog internally starts ncvlog that checks the test fixture file for any syntax error and displays appropriate messages.

Note: The executable and log file names will depend on the simulator being used. For the changes in the executable and log file name when using the Xcelium simulator, see Running Simulations with Xcelium on page 211.

Copying Edit Test Fixture Files

You can copy the current test fixture files to the current or a different run directory. You would typically do this if you want to edit the file for the current simulation, but you want to use it as it is for another simulation run.

To copy the currently selected test fixture file,

1. Click *Copy Selected File* ( ).

   The *Copy To* form appears.

2. Browse to the directory where you want to save a copy of the current file.
3. In the *File name* field, specify the name with which you want to save a copy of the file.
4. Click *Save*.

A copy of the testbench file is created in the specified directory.

Deleting Test Fixture Files

To delete the selected testbench or the stimulus file, click *Delete Selected File* ( ) next to the file name. A dialog box appears prompting you to confirm that you want to delete the file. Click *Yes* to delete the file. The file is removed from the *File Name* field. It is also deleted from the run directory in which it was created.

A
Customizing Your Environment

The information in this appendix describes how to:

- Customizing Command Form Option Settings
- Specifying an Editor for Text Files
- Generating Logical Verilog Netlists of Designs

Customizing Command Form Option Settings

There are two ways to customize command form option settings for the Virtuoso Verilog Environment for the NC-Verilog Integration window (main NC-Verilog window):

- Directly edit the default option settings on the command forms and save the session.
- Create a `.simrc` file that contains the option settings you want to specify.

Directly Editing the Setup Command Forms

When you edit the default settings on a form, the system saves the new settings to the `.vlogifrc` file. The `.vlogifrc` file settings override the default setup options supplied with NC-Verilog Integration Environment or supplied by a user in an `.simrc` file.

The `.vlogifrc` file is rewritten whenever you complete one of the following actions:

- Exit the run directory
- Choose the *File – Quit* command from the NC-Verilog Integration window
- Change to a new run directory

Creating the .simrc File

You create a `.simrc` file with your system editor. Then you add a Cadence® SKILL variable and a custom value (or option setting) to the `.simrc` file.

Form Options and SKILL Variables

In general, you can customize some of the options for each of the following forms.

- SDF Delay Annotation Setup
- Netlist Setup
- Record Signals Setup
- Simulation Setup
- Simulation Comparison Setup
- Cross Selection Setup

The tables in the following section map the form options with the SKILL variables that you can specify in the `.simrc` file.

SDF Delay Annotation Setup Form

Form Option	SKILL Variable
Suppress SDF CellType Checking	simNCVerilogSuppressCellTypeCheck
Suppress sdf2sdf3 Running	simNCVerilogSuppressSdf2Sdf
Suppress sdf2sdf3 Warning Message	simNCVerilogSuppressWarnMessage
Delay File	simNCVerilogSDFFileName
sdf2sdf File	simNCVerilogSdf2SdfOutputFileName
SDF Scope	simNCVerilogSDFScope
SDF Module Instance	simNCVerilogSDFModuleInst
Delay Config File	simNCVerilogSDFConfigFile
Log File	simNCVerilogSDFLogFile
Use Delay Type	simNCVerilogSDFDelayType

Form Option	SKILL Variable
Scale Source Delay	simNCVerilogSDFScaleType
Scale Factor	simNCVerilogSDFScaleFactorConfig
Minimum	simNCVerilogSDFScaleMin
Typical	simNCVerilogSDFScaleTyp
Maximum	simNCVerilogSDFScaleMax

Netlist Setup Form

Form Option	SKILL Variable
Netlisting Mode	simReNetlistAll
Netlist These Views	verilogSimViewList
Netlist For LAI/LMSI Models	simVerilogLaiLmsiNetlisting
Netlist Uppercase	vtoolsUseUpperCaseFlag
Generate Pin Map	hnlVerilogCreatePM
Preserve Buses	simVerilogFlattenBuses
Netlist SwitchRC	simVerilogHandleSwitchRCData
Skip Null Port	simVerilogProcessNullPorts
Netlist Uselib	simVerilogHandleUseLib
Drop Port Range	simVerilogDropPortRange
Incremental Config List	simVerilogIncrementalNetlistConfigList
Symbol Implicit	hnlVerilogNetlistStopCellImplicit
Assign For Alias	vlogifUseAssignsForAlias
Skip Timing Information	vlogifSkipTimingInfo
Declare Global Locally	vlogifDeclareGlobalNetLocal
Netlist Explicitly	simVerilogNetlistExplicit
Support Escape Names	simVerilogEnableEscapeNameMapping
Single Netlist File	simVerilogGenerateSingleNetlistFile

Form Option	SKILL Variable
Terminal SyncUp	hnlVerilogTermSyncUp
Stop Netlisting at Views	verilogSimStopList
Global Power Nets	simVerilogPwrNetList
Global Ground Nets	simVerilogGndNetList
Global TimeScale Overwrite Schematic Time Scale	simVerilogOverWriteSchTimeScale
Global Sim Time	simVerilogSimTimeValue
Unit (Global Sim Time)	simVerilogSimTimeUnit
Global Sim Precision	simVerilogSimPrecisionValue
Unit (Global Precision Time)	simVerilogSimPrecisionUnit

Simulation Options Form

Form Option	SKILL Variable
Options File	simNCVerilogOptFile
Files	simNCVerilogLibFile
Directories	simNCVerilogLibDir
Pack Reference Library In Dir	simNCVerilogPackLib
Pack Reference Library	simNCVerilogPackButton
Exit After	simNCVerilogStepComp
Exit After	simNCVerilogStepElab
Read	simNCVerilogReadAccess
Write	simNCVerilogWriteAccess
Connectivity	simNCVerilogConnectAccess
Enable Line Debugging	simNCVerilogLineDebug
Mode	simNCVerilogDelayMode
Type	simNCVerilogDelayType
SDF Command File	simNCVerilogSDFDFile

Form Option	SKILL Variable
Use Pulse Control Parameters	simNCVerilogPulseCtlError
Use Pulse Control Parameters	simNCVerilogPulseCtlReject
Use Pulse Control Parameters	simNCVerilogPulseCtlSpecparam
Enable Timing Check	simNCVerilogEnableTimingCheck
Allow Negative Values	simNCVerilogTimingNeg
Ignore Notifiers	simNCVerilogTimingNot
Suppress Warnings	simNCVerilogSupWarn
Simulation Log File	simNCVerilogLogFile
NC-Verilog Executable	simNCVerilogSimBinary

NC Sim Compare Options Form

Form Option	SKILL Variable
Rule File	simNCVerilogVCompScriptFile
Golden Database File	simNCVerilogVCompGoldenDbPath
Compare Database File	simNCVerilogVCompCompareDbPath
Start Time	simNCVerilogVCompStartTime
Finish Time	simNCVerilogVCompFinishTime
Time Unit	simNCVerilogVCompCwinTimeUinit
Time Unit	simNCVerilogVCompTolTimeUinit
Time Unit	simNCVerilogVCompSetupHoldTimeUnit
Positive Tolerance	simNCVerilogVCompPosTol
Negative Tolerance	simNCVerilogVCompNegTol
Clock Signal To Use	simNCVerilogVCompClkDef
Sample Time Unit Offset	simNCVerilogVCompClkSample
Clock Setup Time	simNCVerilogVCompClkSetupTime
Clock Hold Time	simNCVerilogVCompClkHoldTime

Form Option	SKILL Variable
Active Clock Edge For Sampling	simNCVerilogVCompClkEdge
Max. Mismatches	simNCVerilogVCompMaxCompareMismatches
Text File	simNCVerilogVCompTextRepString
Report Verbosity Level	simNCVerilogVCompVerboseCyclic
Difference Database	simNCVerilogVCompCompareResultsDirName
Path to Database	simNCVerilogVCompCompareResultsDirPath
No prompt in GUI	simNCVerilogVCompScriptFileUsageCyclic
No prompt in GUI	simNCVerilogVCompCompWindowButton
No prompt in GUI	simNCVerilogVCompClockButton
No prompt in GUI	simNCVerilogVCompGenTextRepButton
No prompt in GUI	simNCVerilogVCompShowInWaveformButton
No prompt in GUI	simNCVerilogVCompMaxMismatchesButton

Record Signals Options Form

Form Option	SKILL Variable
Trace	simNCVerilogTraceMode
Trace these signals	simNCVerilogTraceSigList
In the Scope	simNCVerilogScope
Depth mode	simNCVerilogScopeRadio
Depth value	simNCVerilogDepth

Cross Selection Options Form

Form Option	SKILL Variable
Cross Selection from Schematic Editor to SimVision	simNCVerilogCompToSyn

Form Option	SKILL Variable
Cross Selection from SimVision to Schematic Editor	simNCVerilogSynToComp

In addition, there is another SKILL variable, `simSupportDuplicatePorts`, which determines whether to remove duplicate ports from a netlist. The variable is set to *t,* by default and the simulator accepts duplicate ports. If set to *nil*, the netlister removes duplicate ports from a netlist.

Specifying an Editor for Text Files

In the Virtuoso design environment, the default editor for text files in Virtuoso Text Editor. For details on this editor, see *Virtuoso Text Editor User Guide*.

You can also specify an editor of your choice in any of the following ways for working with text files, such as Verilog or Verilog-A files:

- Set the SKILL variable `editor` in Virtuoso CIW to specify the text editor as follows:

  ```
  editor="s_editorCommand"
  ```

  Here, *`s_editorCommand`* is the command for opening the required editor. For example, to specify vi as the editor, set this variable as follows:

  ```
  editor="vi"
  ```

- Set the SKILL variable `hdlReadOnlyModeEditorCommand` in CIW or in the `.cdsinit` file to open a file in an external editor in read-only mode:

  ```
  hdlReadOnlyModeEditorCommand="s_readOnlyCommand"
  ```

  Here, `s_readOnlyCommand` is the command for opening files in the read-only mode in the required editor. For example, to open a file in nedit in the read-only mode, set this variable as follows:

  ```
  hdlReadOnlyModeEditorCommand="nedit -read"
  ```

Generating Logical Verilog Netlists of Designs

This section includes the following topics:

- Overview
- Terminology
- Generating a Logical Verilog Netlist
- Comparison Between a Regular and a Logical Verilog Netlist

Overview

Consider a scenario where you want to run low-power simulations on a Verilog design that contains well-defined power connections along with inherited and explicit connections. For running these simulations, you would supply as input to the simulator, a netlist of the Verilog design. Moreover, the netlist for such a design would contain both connectivity and power information. However, if you are using a simulator such as Innovus™ Digital Implementation System (Innovus), you need to supply as input a netlist that contains only connectivity information, because this simulator obtains the power information from a separate source file. For such simulators, you can generate a logical Verilog netlist.

A logical Verilog netlist contains only connectivity information. It has the following properties:

- Does not contain any inherited connection information, such as:
  ```
  (* netExpr = "vdde(vdde_)" *) wire vdde_;
  (* netExpr = "vdd(vdd_)" *) wire vdd_;
  ```
- Does not list any power and ground pins, and power and ground nets
- Replaces the logical port connections to power and ground nets with `1’b1` or `1’b0`

When generating a logical Verilog netlist, the netlister performs the following actions:

- Power information, that is power or ground terminals and `instTerms`, is removed.
- Ports with signal type other than `power`, `ground`, `tieHi`, or `tieLo` and connected to a net with signal type `ground`, are replaced with `tieLo` or `1'b0`.
- Ports with signal type other than `power`, `ground`, `tieHi`, or `tieLo` and connected to a net with signal type `power`, are replaced with `tieHi` or `1'b1`.

For information on the terms used to define a logical Verilog netlist, see “Terminology” on page 202.

To generate a logical Verilog netlist, set the SKILL variable `simVerilogGenerateLogicalVerilog` and netlist the design using NC-Verilog Environment.

Note: The `simVerilogGenerateLogicalVerilog` SKILL variable is not supported in SystemVerilog.

If you do not want to explicitly specify the connections as scalar constants `1'b1` and `1'b0`, you can declare the power and ground connections as `supply1` and `supply0` respectively in the logical Verilog netlist using the SKILL variable `hnlVerilogLogicalWithSupplies`. For example, the use of this variable is recommended in a design hierarchy with instances of primitive gates and their supply nets resolved with `1'b1` or `1'b0`, which causes the elaboration process to fail. To elaborate the design successfully, set the variable to generate a logical Verilog netlist where the power and ground connections are declared as `supply1` and `supply0`.

For details on how to generate these types of logical Verilog netlists of designs, see "Generating a Logical Verilog Netlist" on page 203.

The following box illustrates a logical Verilog netlist where the logical port connections to power and ground nets are replaced by the corresponding scalar constant through the use of `simVerilogGenerateLogicalVerilog`.

```
`timescale 1ns / 1ns
module BOTTOM ( Z, A );
output  Z;
input  A;
specify
    specparam CDS_LIBNAME  = "testlib";
    specparam CDS_CELLNAME = "BOTTOM";
    specparam CDS_VIEWNAME = "schematic";
endspecify
tranif1 N0( Z, 1'b0, A);
tranif0 P0( Z, 1'b1, A);
endmodule
```

The following box illustrates a logical Verilog netlist where the supply nets are declared as `supply1` or `supply0` through the use of `hnlVerilogLogicalWithSupplies` and `simVerilogGenerateLogicalVerilog`.

```
`timescale 1ns / 1ns
module BOTTOM ( Z, A );
output  Z;
input  A;
// Supply declaration
supply1  VDD;
supply0  VSS;
specify
    specparam CDS_LIBNAME  = "testlib";
    specparam CDS_CELLNAME = "BOTTOM";
    specparam CDS_VIEWNAME = "schematic";
endspecify
// VDD and VSS retained in the netlist and not replaced
tranif1 N0( Z, VSS, A);
tranif0 P0( Z, VDD, A);
endmodule
```

Terminology

The table shown below describes the terms used to define a logical Verilog netlist. The definitions of these terms are based on the value of the signal type *(*`sigType`*)* attribute that you set for a net or terminal in your Verilog design. Virtuoso uses the value of `sigType` to extract power and ground nets, and the `tieHi` and `tieLo` connections from the design.

You can set `sigType` of a net or terminal in Virtuoso Schematic Editor or in Virtuoso Symbol Editor by using one of the following:

- Edit Object Properties form–in the *Signal Type* list box
- Property Editor assistant

For information about the *Signal Type* list box, see “Add Pin Form - Schematic” in *Virtuoso Schematic Editor L User Guide*.

Term	Definition
`Power net`	A net that has the signal type attribute set to `power`

`Ground net`	A net that has the signal type attribute set to `ground`
`Logical net`	A net that has the signal type attribute set to any value other than `power`, `tieHi`, `ground`, or `tieLo`
`Power terminal`	A terminal (schematic pin) connected to a net that has the signal type attribute set to `power`
`Ground terminal`	A terminal (schematic pin) connected to a net that has the signal type attribute set to `ground`
`Logical terminal`	A terminal (schematic pin) connected to a logical net or to nets that have the signal type attribute set to `tieHi` or `tieLo`
`Power instTerm`	An instTerm for which the corresponding terminal has the signal type attribute set to `power`
`Ground instTerm`	An instTerm for which the corresponding terminal has the signal type attribute set to `ground`
`Logical instTerm`	An instTerm for which corresponding terminal is connected to a logical net or that has the signal type attribute set to `tieHi` or `tieLo`

Generating a Logical Verilog Netlist

To generate a logical Verilog netlist of a design where the logical port connections to power and ground nets are replaced with the corresponding scalar constant, set the SKILL variable `simVerilogGenerateLogicalVerilog` to `t` and identify the power and ground nets in the design. If you want to generate the logical Verilog netlist where the connections are declared as `supply0` and `supply1` appropriately, set the SKILL variable `hnlVerilogLogicalWithSupplies` to `t`, in addition to setting `simVerilogGenerateLogicalVerilog`. For details and examples of these types of logical Verilog netlists, see “Overview” on page 200.

You can identify the power and ground nets using one of the following methods:

- Setting variables
- Editing the design

You can then generate a logical netlist of the design from NC-Verilog Integration Environment.

Generating a Logical Verilog Netlist by Setting Variables

You can generate a logical Verilog netlist of a design by setting the applicable environment variables. This method eliminates the need to modify the design to indicate the power and ground nets. It lets you generate a logical netlist where cellview terminals identified through a variable are ignored in the logical netlist.

To generate a logical Verilog netlist of a design:

1. Set the `simVerilogGenerateLogicalVerilog` variable to `t`. If required, set `hnlVerilogLogicalWithSupplies` to `t`.

   You can set the variables in `.simrc`, `si.env`, or Virtuoso CIW.

2. Set the netlister to ignore the power and ground terminals in the design by specifying the `hnlVerilogIgnoreTermNameList` variable in `.simrc`, `si.env`, or Virtuoso CIW. To set this variable, specify the power and ground terminals to be ignored in the netlist in the following syntax:

   ```
   hnlVerilogIgnoreTermNameList=(list "ignoreTermName1" "ignoreTermName2" ...)
   ```

   Note: The `hnlVerilogIgnoreTermNameList` variable supports wild cards for specifying the power and ground terminal names. For example,
   `hnlVerilogIgnoreTermNameList = list("vcc*" "vss*")`
   The generated Verilog netlist ignores all names that include the specified list value.

3. Set the netlister to identify the global power and ground nets in the design. For this, do one of the following tasks:

   ❑ Set the simVerilogPwrNetList and simVerilogGndNetList variables in `.simrc`, `si.env`, or Virtuoso CIW in the following syntax:

   ```
   simVerilogPwrNetList=(list "powerNetName1" "powerNetName2" ... )
   simVerilogGndNetList=(list "groundNetName1" "groundNetName2" ... )
   ```

   ❑ Perform the following steps to set the global power and ground nets from the graphical user interface:

   a. Launch the NC-Verilog Integration Environment for the design.

   b. Initialize the run directory.

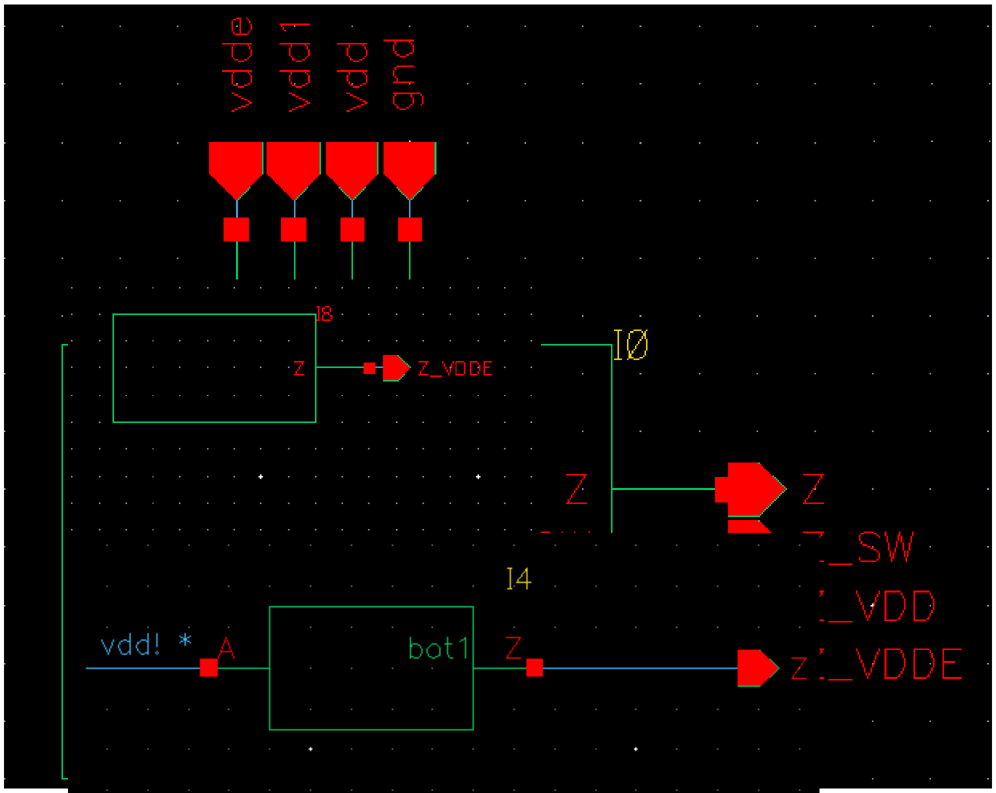
   c. Set the global power and ground signals in the Global Power Nets and Global Ground Nets fields of the Netlist Setup form.

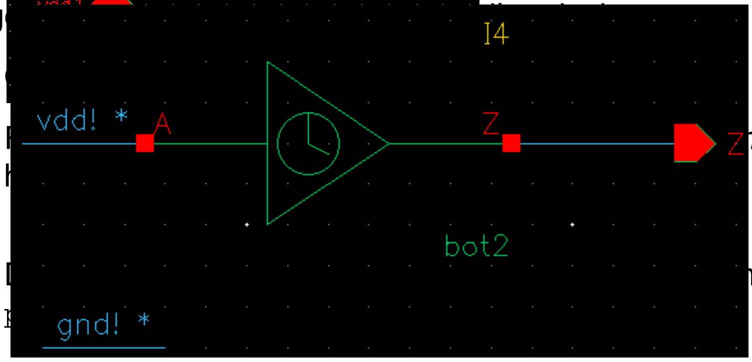
   For details, see Chapter 3, “Setting Up the Simulation Environment.”

4. Netlist the design using the NC-Verilog Integration Environment. For details, see “Netlisting a Design” on page 68.

Generating a Logical Verilog Netlist by Editing the Design

The following figure shows a Verilog design that uses terminals with inherited connections for power and ground supplies.





For t erate two versions of the Verilog
netlis st and a logical netlist. This section
desc ate a regular netlist, see the steps
desc age 68. A comparison of the two netlists, regular and
logic sign is shown at the end of this section.

To g

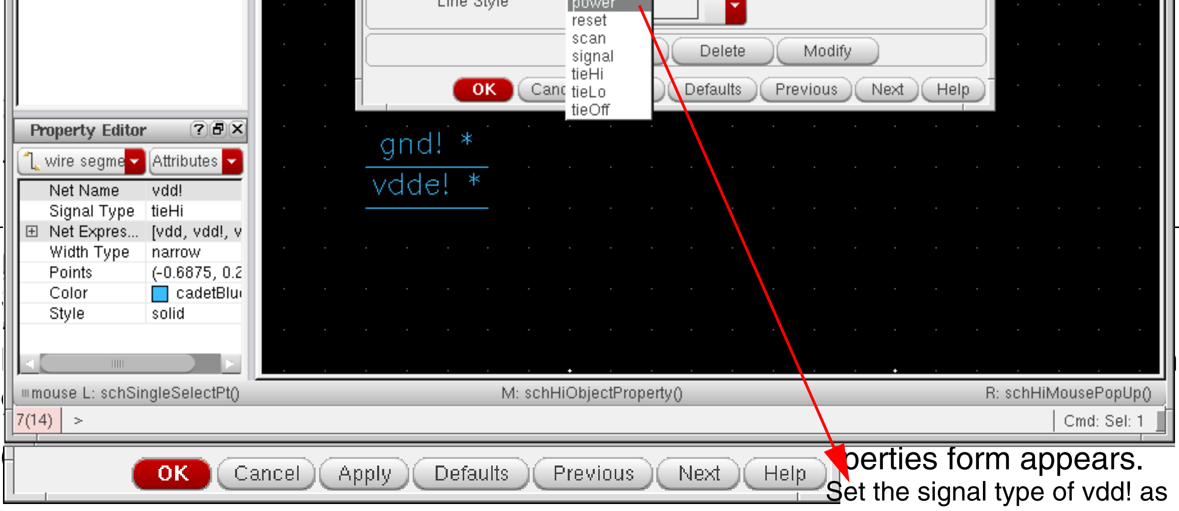
1. (

2. F *nd Edit* to descend into the design
h

D nt to set the signal type attribute as
I

3. ets

4. perties form appears.



Net Name	vdd!
Signal Type	tieHi
⊞ Net Expres...	[vdd, vdd!, v
Width Type	narrow
Points	(-0.6875, 0.2
Color	🟦 cadetBlu
Style	solid

mouse L: schSingleSelectPt() M: schHiObjectProperty() R: schHiMousePopUp()

7(14) > Cmd: Sel: 1

Set the signal type of vdd! as `power` from the Signal Type list box.

5. From the *Signal Type* list box, select `power`.

   Note: You can also modify the signal type attribute by using the Property Editor assistant.

6. Repeat steps 3 to 5 listed above for the following components:

   ❑ Nets for which you want to set the signal type attribute as `ground`.

   Note: Ensure that the nets that are neither power nor ground are set to a signal type that is different from `power`, `ground`, `tieHi`, or `tieLo`.

   ❑ Terminals for which you want to set the signal type attribute as `power` and `ground`.

   ❑ Signals for which you want to set the signal type attribute as `tieHi` and `tieLo`.

7. Set the `simVerilogGenerateLogicalVerilog` variable to `t`. If required, set `hnlVerilogLogicalWithSupplies` to `t`.

   You can set the variables in `.simrc`, `si.env`, or Virtuoso CIW. For details on these variables, see "Generating a Logical Verilog Netlist" on page 203.

8. Launch the NC-Verilog Integration Environment for the design. For this, choose *Launch — Plugin — Simulation — NC-Verilog*.

9. Initialize the run directory.

10. Set the global power and ground signals in the Global Power Nets and Global Ground Nets fields of the Netlist Setup form.

    For details, see Chapter 3, "Setting Up the Simulation Environment."

11. Generate the netlist. For instructions, see "Netlisting a Design" on page 68.

You can now view the netlist to check if it is the required logical Verilog netlist. To view the netlist, choose *Results – Netlist*.

Comparison Between a Regular and a Logical Verilog Netlist

A regular and a logical Verilog netlist generated from the Verilog design illustrated in "Generating a Logical Verilog Netlist by Editing the Design" on page 205 are given in the table below for comparison.

Notice the following in the logical netlist shown:

- Power information, that is power or ground terminals and `instTerms`, is removed.
- Ports with signal type other than `power`, `ground`, `tieHi`, or `tieLo` and connected to a net with signal type `ground`, are replaced with `tieLo` or `1'b0`.
- Ports with signal type other than `power`, `ground`, `tieHi`, or `tieLo` and connected to a net with signal type `power`, are replaced with `tieHi` or `1'b1`.

Regular Netlist	Logical Netlist

`// Library - TEST_MyCAD, Cell - mycell,`
`View - schematic`
`// LAST TIME SAVED: Jun 20 16:50:00 2011`
`// NETLIST TIME: Jun 21 21:59:35 2011`
`` `timescale 1ns / 1ns ``
 	
`module mycell ( Z, Z_SW, Z_VDD, Z_VDDE,`
`gnd, vdd, vdd1, vdde );`
`output  Z, Z_SW, Z_VDD, Z_VDDE;`
`input  gnd, vdd, vdd1, vdde;`
 	
`top I0 ( Z, Z_SW, Z_VDD, Z_VDDE, gnd, vdd,`
`vdd1, vdde);`
`endmodule`
	`// Library - TEST_MyCAD, Cell - mycell,`
	`View - schematic`
	`// LAST TIME SAVED: Jun 20 16:50:00`
	`2011`
	`// NETLIST TIME: Jun 21 21:51:21 2011`
	`` `timescale 1ns / 1ns ``
	
	`module mycell ( Z, Z_SW, Z_VDD, Z_VDDE,`
	`gnd, vdde );`
	`output  Z, Z_SW, Z_VDD, Z_VDDE;`
	`input  gnd, vdd;`
	
	`top I0 ( Z, Z_SW, Z_VDD, Z_VDDE, );`
	`endmodule`

`// Library - TEST_MyCAD, Cell - top, View`
`- schematic`
`// LAST TIME SAVED: Jun 20 16:15:30 2011`
`// NETLIST TIME: Jun 21 21:59:34 2011`
`` `timescale 1ns / 1ns ``
 	
`module top ( Z, Z_SW, Z_VDD, Z_VDDE, gnd,`
`vdd, vdd1, vdde );`
`output  Z, Z_SW, Z_VDD, Z_VDDE;`
`inout  gnd, vdd, vdd1, vdde;`
`mid1 I7 ( Z_VDD, gnd, vdd1, vdde);`
`mid1 I8 ( Z_VDDE, gnd, vdd, vdde);`
`mid2 I9 ( Z_SW, gnd, vdd1);`
`mid2 I10 ( Z, gnd, vdd);`
`endmodule`
	`// Library - TEST_MyCAD, Cell - top,`
	`View - schematic`
	`// LAST TIME SAVED: Jun 20 16:15:30`
	`2011`
	`// NETLIST TIME: Jun 21 21:51:20 2011`
	`` `timescale 1ns / 1ns ``
	`module top ( Z, Z_SW, Z_VDD, Z_VDDE );`
	`output  Z, Z_SW, Z_VDD, Z_VDDE;`
	
	`mid1 I7 ( Z_VDD);`
	`mid1 I8 ( Z_VDDE);`
	`mid2 I9 ( Z_SW);`
	`mid2 I10 ( Z);`
	`endmodule`

`// Library - TEST_MyCAD, Cell - mid1, View`
`- schematic`
`// LAST TIME SAVED: Jun 20 16:16:06 2011`
`// NETLIST TIME: Jun 21 21:59:34 2011`
`` `timescale 1ns / 1ns ``
 	
`module mid1 ( Z, inh_gnd, inh_vdd,`
`inh_vdde );`
`output  Z;`
`inout  inh_gnd, inh_vdd, inh_vdde;`
`LEVELSHIFTER I4 ( Z, inh_vdd);`
`endmodule`
	`// Library - TEST_MyCAD, Cell - mid1,`
	`View - schematic`
	`// LAST TIME SAVED: Jun 20 16:16:06`
	`2011`
	`// NETLIST TIME: Jun 21 21:51:20 2011`
	`` `timescale 1ns / 1ns ``
	
	`module mid1 ( Z );`
	`output  Z;`
	`LEVELSHIFTER I4 ( Z, 1'b1);`
	`endmodule`

Regular Netlist	Logical Netlist
`// Library - TEST_MyCAD, Cell - mid2, View - schematic`	`// Library - TEST_MyCAD, Cell - mid2, View - schematic`
`// LAST TIME SAVED: Jun 20 16:17:06 2011`	`// LAST TIME SAVED: Jun 20 16:17:06 2011`
`// NETLIST TIME: Jun 21 21:59:34 2011`	`// NETLIST TIME: Jun 21 21:51:20 2011`
`` `timescale 1ns / 1ns ``	`` `timescale 1ns / 1ns ``
`module mid2 ( Z, inh_gnd, inh_vdd );`	`module mid2 ( Z );`
`output  Z;`	`output  Z;`
`inout  inh_gnd, inh_vdd;`	
`CNBF I4 ( Z, inh_vdd);`	`CNBF I4 ( Z, 1'b1);`
`endmodule`	`endmodule`

B
Running Simulations with Xcelium

You can also simulate and debug your designs using the Xcelium simulator. With this simulator in place, all the executable names, log file names, and output directory names have been changed. However, the old executable and log file names will continue to work for other simulators.

The following table lists the changes in the executable and log file names when using the Xcelium simulator:

Old Executable	Old log file name	New Executable	New log file name
irun	irun.log	xrun	xrun.log
iprof	iprof.log	xprof	xprof.log
ncsim	ncsim.log	xmsim	xmsim.log
ncelab	ncelab.log	xmelab	xmelab.log
ncvlog	ncvlog.log	xmvlog	xmvlog.log
ncvlog_cg		xmvlog_cg	
ncvhdl	ncvhdl.log	xmvhdl	xmvhdl.log
ncvhdl_cg		xmvhdl_cg	
ncsc	ncsc.log	xmsc	xmsc.log
ncsc_run	ncsc.log	xmsc_run	xmsc_run.log
ncls	ncls.log	xmls	xmls.log
nchelp	nchelp.log	xmhelp	xmhelp.log
ncdc	ncdc.log	xmdc	xmdc.log
ncverilog	ncverilog.log	xmverilog	xmverilog.log
ncprep	ncprep.log	xmprep	xmprep.log