

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

**Product Version IC23.1
June 2023**

© 2023 Cadence Design Systems, Inc. All rights reserved.
Printed in the United States of America.

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

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522. All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Patents: The Cadence Products covered in this manual are protected by U.S. Patents

5,790,436; 5,812,431; 5,859,785; 5,949,992; 6,493,849; 6,278,964; 6,300,765; 6,304,097; 6,414,498; 6,560,755; 6,618,837; 6,693,439; 6,826,736; 6,851,097; 6,711,725; 6,832,358; 6,874,133; 6,918,102; 6,954,908; 6,957,400; 7,003,745; 7,003,749.

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

Cadence is committed to using respectful language in our code and communications. We are also active in the removal and replacement of inappropriate language from existing content. This product documentation may however contain material that is no longer considered appropriate but still reflects long-standing industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

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

Contents

1

<u>Introducing SystemVerilog Integration Environment</u>	5
<u>Licensing Requirements</u>	6
<u>Key Features</u>	6
<u>Netlist Generation and Design Simulation Flow</u>	8
<u>Tool Requirements</u>	10
<u>Graphical User Interface</u>	10
<u>Launching the Graphical User Interface</u>	11
<u>Understanding the Graphical User Interface</u>	12
<u>About Creating SystemVerilog-Based Designs</u>	14

2

<u>Generating Netlist and Simulating Designs</u>	21
<u>Overview</u>	22
<u>Initializing the Run Directory</u>	23
<u>Netlisting a Design</u>	26
<u>Configuring Options for Generating a Netlist</u>	26
<u>Printing CDF Parameters in Inline Explicit Format</u>	34
<u>Managing Data Type Conflicts</u>	35
<u>Overriding Hierarchical Data Type Propagation</u>	35
<u>Ignoring Port Type Propagation</u>	36
<u>Adding Port Properties to an Instance</u>	37
<u>Generating a Netlist</u>	42
<u>Viewing a Netlist and a Map File</u>	43
<u>Simulating a Design</u>	44
<u>Configuring Options for Simulating a Design</u>	45
<u>Specifying the Testbench File and Stimulus File</u>	51
<u>Simulating a Design in Interactive Mode</u>	53
<u>Simulating a Design in Batch Mode</u>	55
<u>Using Standalone Mode</u>	57

A

<u>Configuration Flags</u>	59
<u>Configuration Flags for Design Details</u>	60
<u>Configuring Flags for Netlist Generation</u>	60
<u>Configuring Flags for Simulation</u>	62
<u>Configuring Flags for Test Fixture Files</u>	63
<u>Other Flags for Netlist Configuration</u>	63

B

<u>Examples</u>	67
<u>Overriding Hierarchical Data Type Propagation in a Design</u>	68
<u>Resolving Data Type Conflict</u>	70
<u>Netlisting a Design Containing Packed and Unpacked Arrays</u>	73

C

<u>Running Simulations with Xcelium</u>	75
---	----

Introducing SystemVerilog Integration Environment

Virtuoso® Verilog Environment for SystemVerilog Integration (SystemVerilog Integration Environment) is an environment for generating netlists of SystemVerilog-based digital designs. This environment also integrates with other Cadence tools to simulate and debug designs.

This topic describes how to use the Virtuoso® Verilog® Environment for SystemVerilog Integration (SystemVerilog Integration Environment) to netlist and simulate SystemVerilog-based designs.

This topic is aimed at developers and designers of integrated 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, the Virtuoso Layout Suite, and Virtuoso Schematic Editor.
- The Virtuoso Studio design environment technology file.
- Component description format (CDF), which lets you create and describe your own components for use with Layout XL.

This document includes the following topics.

- [Licensing Requirements](#) on page 6
- [Key Features](#) on page 6
- [Netlist Generation and Design Simulation Flow](#) on page 8
- [Tool Requirements](#) on page 10
- [Graphical User Interface](#) on page 10
- [About Creating SystemVerilog-Based Designs](#) on page 14

Licensing Requirements

SystemVerilog Integration Environment requires the following licenses:

- Virtuoso Schematic Editor XL license (License Number 95115)
- Virtuoso Schematic Editor Verilog Interface license (License Number 21400)

For information about licensing in the Virtuoso Studio design environment, see [*Virtuoso Software Licensing and Configuration Guide*](#).

Key Features

SystemVerilog Integration Environment provides a methodology to netlist and simulate SystemVerilog-based designs. This methodology includes the following stages:

1. Initialize the run directory for storing the netlist and simulation data.
2. Generate the netlist of the design that describes the connectivity of the design.
3. Simulate the design using the generated netlist as an input.

SystemVerilog Integration Environment provides the following key features and capabilities:

- A graphical user interface and a command line interface
- Various options for configuring netlist generation and design simulation
- Support for various SystemVerilog data types, such as release interface, custom, wire, reg, logic, string, and event
- Data type progression from leaf-level SystemVerilog cellviews to top-level schematic design hierarchy, and an option to override the progression
- Netlist generation for configuration and non-configuration based designs
- Netlist generation and simulation of mixed-language design where Verilog and SystemVerilog views coexist at the leaf level
- Support for handling packed and unpacked arrays in the design during netlist generation
- Awareness of the following file extensions:
 - ❑ SystemVerilog file extensions `.sv`, `.SV`, `.svp`, `.SVP`, `.svi`, `.svh`, `.vlib`, `.VLIB`
 - ❑ Verilog file extensions `.v`, `.V`, `.vp`, `.VP`, `.vs`, and `.VS`
 - ❑ Other user-specified Verilog and SystemVerilog file extensions

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

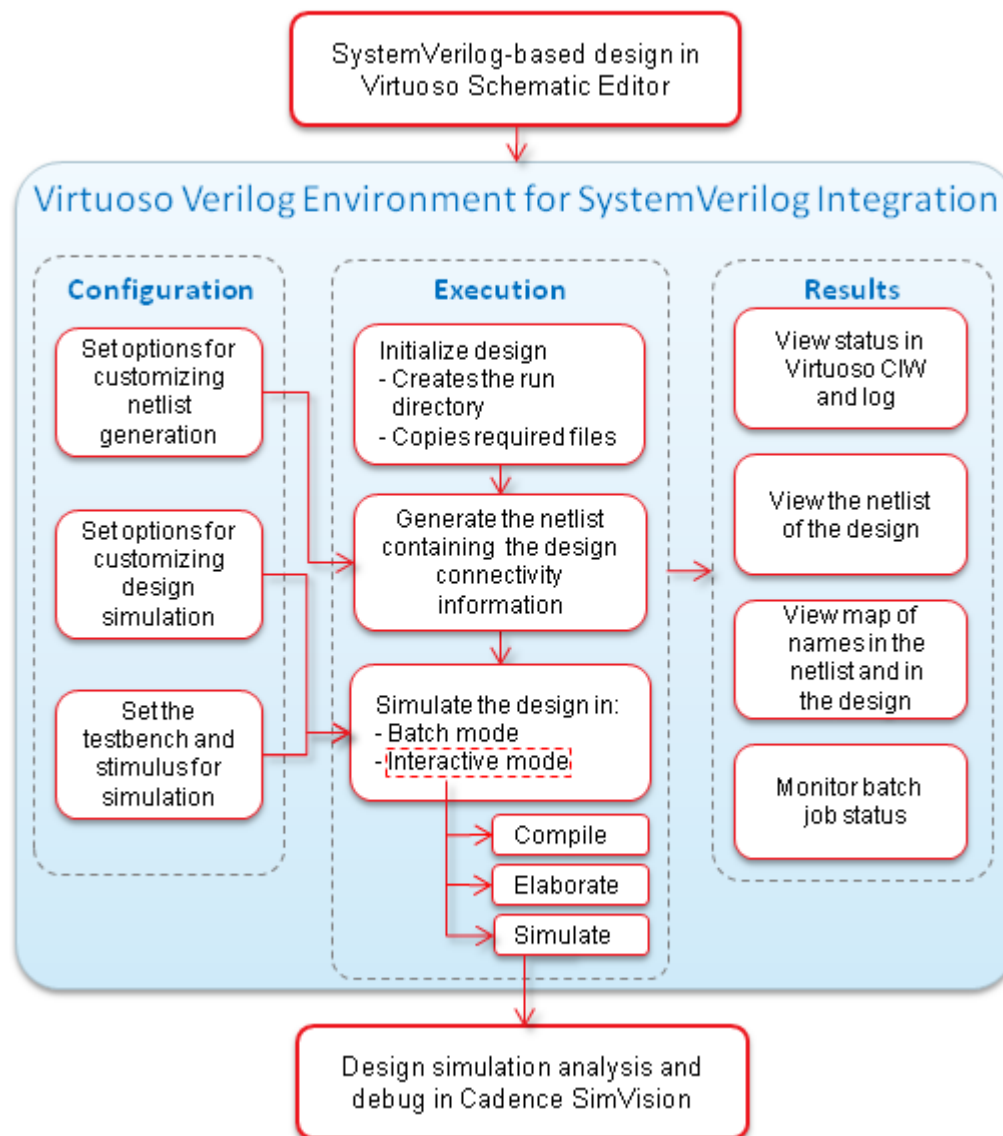
Introducing SystemVerilog Integration Environment

- Design simulation in the following modes:
 - Interactive mode that lets you choose the compilation, elaboration, and simulation steps to perform
 - Batch mode that performs all steps
- Integration with Cadence® SimVision to let you simulate and debug designs in interactive mode

Note: The Cadence® SimVision is a simulation analysis environment for debugging digital, analog, or mixed-signal designs written in Verilog, SystemVerilog, VHDL, SystemC, or a combination of those languages. For more information, see the *SimVision User Guide*.

Netlist Generation and Design Simulation Flow

The following figure illustrates the flow through which SystemVerilog Integration Environment generates a netlist of a design and simulates that design.



The generic steps for using SystemVerilog Integration Environment are as follows:

1. Create your design. The design can have Verilog and SystemVerilog cellviews coexisting at the leaf level.

Note: Virtuoso lets you create a SystemVerilog text-only cellview and a symbol for it, which you can integrate in your design. For details, see [“About Creating SystemVerilog-Based Designs”](#) on page 14.

2. Open the top cellview of the design in Virtuoso Schematic Editor L or XL and launch SystemVerilog Integration Environment. You can open the schematic view or the configuration view of the design.

For details, see [“Launching the Graphical User Interface”](#) on page 11.

3. Initialize the run directory for netlist generation and design simulation.

For details, see [“Initializing the Run Directory”](#) on page 23.

4. Set options for netlist generation and generate the netlist of the design.

SystemVerilog Integration Environment also creates a map file containing:

- ☐ The netlist configuration options.
- ☐ Map of the names used in the netlist and their corresponding name in the design.

You can view contents of the netlist and the map file.

For details, see [“Netlisting a Design”](#) on page 26.

5. Set options for simulating the design. You can also set the testbench and stimulus. Then choose the batch or interactive mode and start the simulation.

If you choose the interactive mode, SystemVerilog Integration Environment launches SimVision to simulate the design after the design compilation and elaboration is complete.

If you choose the batch mode, the environment simulates the design in the background. The environment provides a window to monitor the status of design simulation in the batch mode.

For details, see [“Simulating a Design”](#) on page 44.

6. Analyze simulation results as required.

If you encounter issues using SystemVerilog Integration Environment, refer to the log that the environment displays in the Virtuoso CIW. It also displays the status of the last operation on the main form.

Tool Requirements

You launch SystemVerilog Integration Environment from the Virtuoso Schematic Editor L or XL.

To perform its operations, SystemVerilog Integration Environment requires the following tools:

- Cadence `xrun` utility

The `xrun` utility is used to simulate a design. It specifies all the input files and options in a single command line.

The `xrun` utility can take SystemVerilog and Verilog files as inputs. The utility uses the Native Code tools to compile, elaborate, and simulate the design.

Note: The executable and log file names will depend on the simulator being used. For the changes in the executable and log file names when using the Xcelium simulator, see [Running Simulations with Xcelium](#) on page 75

- Cadence Native Code (NC) tools

The `xrun` utility uses the following NC tools to simulate designs:

- ❑ Native Code Compiler (`xmvlog`) compiles SystemVerilog and Verilog-based designs.
- ❑ Native Code Elaborator (`xmelab`) gathers various portions of a design and creates a snapshot for simulation.
- ❑ Native Code Simulator (`xmsim`) simulates the design in the batch mode.

Note: The `xrun` utility uses the input filename extension to determine the compiler it should use to compile the input files. To compile SystemVerilog files, the utility runs `xmvlog` with the `-sv` option.

- Cadence SimVision

SimVision provides an environment for simulating and debugging digital designs written in SystemVerilog or a mix of Verilog and SystemVerilog languages. SystemVerilog Integration Environment launches this tool after design compilation and elaboration is completed in the interactive mode.

Graphical User Interface

You can use SystemVerilog Integration Environment from its graphical user interface and command line interface.

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Introducing SystemVerilog Integration Environment

This section provides information on the following topics:

- Launching the Graphical User Interface
- Understanding the Graphical User Interface

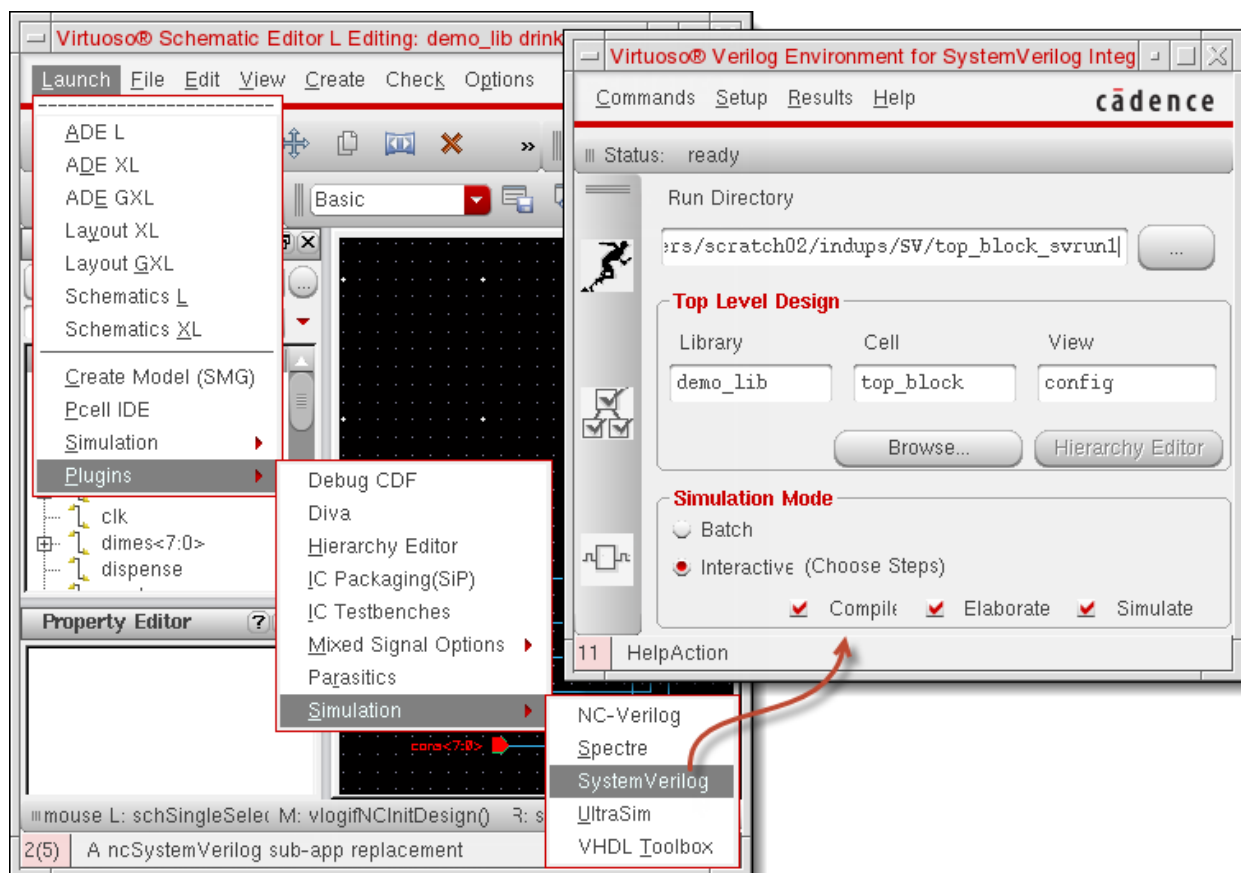
For information on the command line interface, see “Using Standalone Mode” on page 57.

Note: SystemVerilog Integration Environment and the Virtuoso® Verilog Environment for NC-Verilog Integration have a similar user interface.

Launching the Graphical User Interface

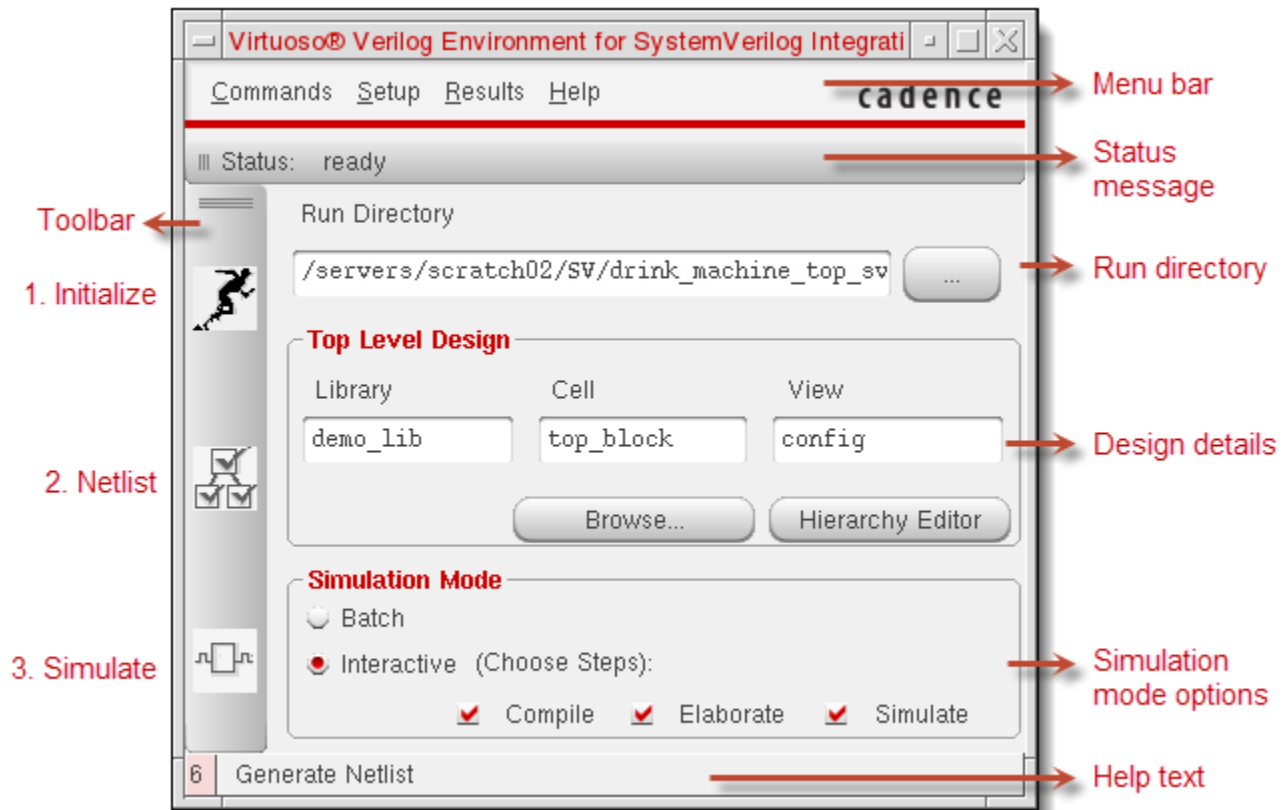
To launch the graphical user interface of SystemVerilog Integration Environment:

1. Open the top cell of your SystemVerilog-based design in Virtuoso Schematic Editor L or XL. You can open the top cell using the schematic view or the configuration view.
2. Choose *Launch — Plugins — Simulation — SystemVerilog*. The main form of SystemVerilog Integration Environment appears.



Understanding the Graphical User Interface

The following figure illustrates the main form of SystemVerilog Integration Environment.



The following table describes the main form components.

Component	Description
Menu bar	<p>Access the following menus to perform various operations.</p> <ul style="list-style-type: none"> ■ <i>Commands</i>: Access options to initialize the run directory, generate the netlist, specify the testbench and stimulus, and simulate the design. ■ <i>Setup</i>: Access the forms for setting netlist generation options and simulation options. ■ <i>Results</i>: Access options to view the netlist and map. Also access the job monitoring window to view the batch simulation status. ■ <i>Help</i>: Access help on using SystemVerilog Integration Environment.

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Introducing SystemVerilog Integration Environment

Component	Description
Toolbar	<p>Click the following shortcut toolbar option to perform tasks.</p> <ul style="list-style-type: none">■ <i>Initialize Design</i>: Initialize the run directory for netlist generation and design simulation.■ <i>Generate Netlist</i>: Generate a hierarchical netlist of the design in the run directory using the netlist generation settings. This option becomes available after you initialize the run directory.■ <i>Simulate</i>: Simulates the design in interactive or batch mode using the simulation settings. This option becomes available after you generate a netlist of the design.
Status message	<p>Shows the status of the last operation. The status can be:</p> <ul style="list-style-type: none">■ <i>Uninitialized</i>: The run directory must be initialized before you can netlist and simulate the design.■ <i>Ready</i>: The run directory has been initialized and you can start generating the netlist.■ <i>Netlisting Succeeded</i>: The netlist has been generated and you can start simulating the design.■ <i>Compiling Design</i>: Design compilation is in progress.■ <i>Compilation Successful</i>: The design files have been compiled.■ <i>Elaborating Design</i>: Design elaboration is in progress.■ <i>Elaboration Successful</i>: The design has been elaborated and you can start the simulation.■ <i>SimVision Launched</i>: SystemVerilog Integration Environment has launched SimVision for interactive design simulation.■ <i>Batch Simulation</i>: SystemVerilog Integration Environment is simulating or has simulated the design in the batch mode using the xmsim tool.
Run directory	<p>Specify the run directory for storing the netlist, simulation, and waveform data.</p> <p>For details, see “Initializing the Run Directory” on page 23.</p>

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Introducing SystemVerilog Integration Environment

Component	Description
Design details	<p>Specify the library, cell, and view of the top-level design. By default, the environment displays the library, cell, and view of the design opened in Virtuoso Schematic Editor, from where you launched the environment. You can change this design reference.</p> <p>For details, see “Initializing the Run Directory” on page 23.</p>
Simulation mode options	<p>Specify the simulation mode. You can select <i>Interactive</i> or <i>Batch</i> mode of simulation.</p> <p>For details, see “Simulating a Design” on page 44.</p>
Help text	Displays help tips on the selected user-interface component.

About Creating SystemVerilog-Based Designs

Virtuoso lets you create a SystemVerilog text-only cellview and a symbol for this cellview, which you can use in your SystemVerilog-based design. You can create and manage your design using the Virtuoso Schematic Editor, and netlist and simulate this design using SystemVerilog Integration Environment.

For information on working with designs, see the [Virtuoso Schematic Editor L User Guide](#).

Note: Virtuoso provides support for handling packed and unpacked arrays during SystemVerilog symbol generation.

Note: You can create a SystemVerilog cellview and its symbol using the procedure described below. You can import text cellviews from 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.

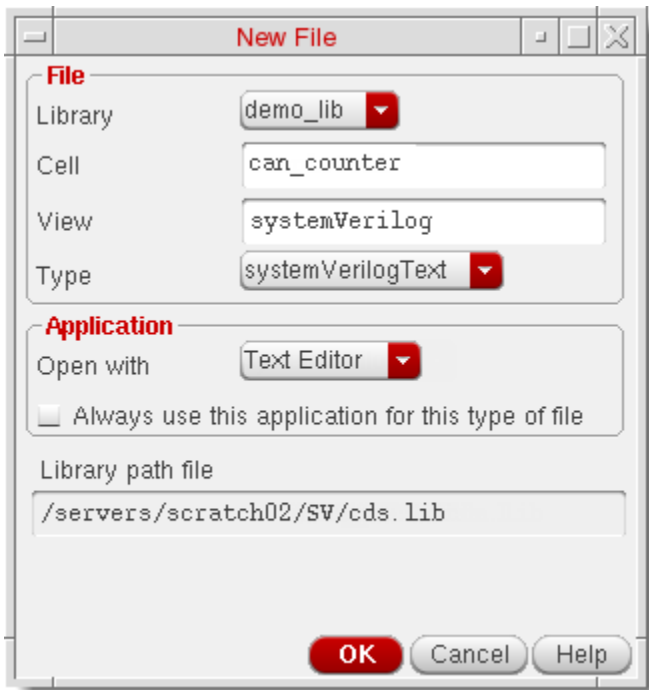
You can also generate the text database for a given cellview without opening the Text Editor, using the [hdlGenerateTextDatabase](#) SKILL function:

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Introducing SystemVerilog Integration Environment

To create a SystemVerilog cellview and symbol:

1. Choose *File* — *New* — *Cellview* from the Virtuoso menu. The New File form appears.



2. Specify the cellview details.

Field	Details
<i>Library</i>	Select the library to which you want to add the new cell.
<i>Cell</i>	Enter the name of the new cell.
<i>View</i>	Specify the default view as required. If you leave this field blank and select a view type from the <i>Type</i> list, the field automatically displays the default view of the selected type,
<i>Type</i>	Select <code>systemVerilogText</code> from the list of view types. If the <i>View</i> field is blank, it updates to display <code>systemVerilog</code> .
<i>Open with</i>	Select the default Text Editor.

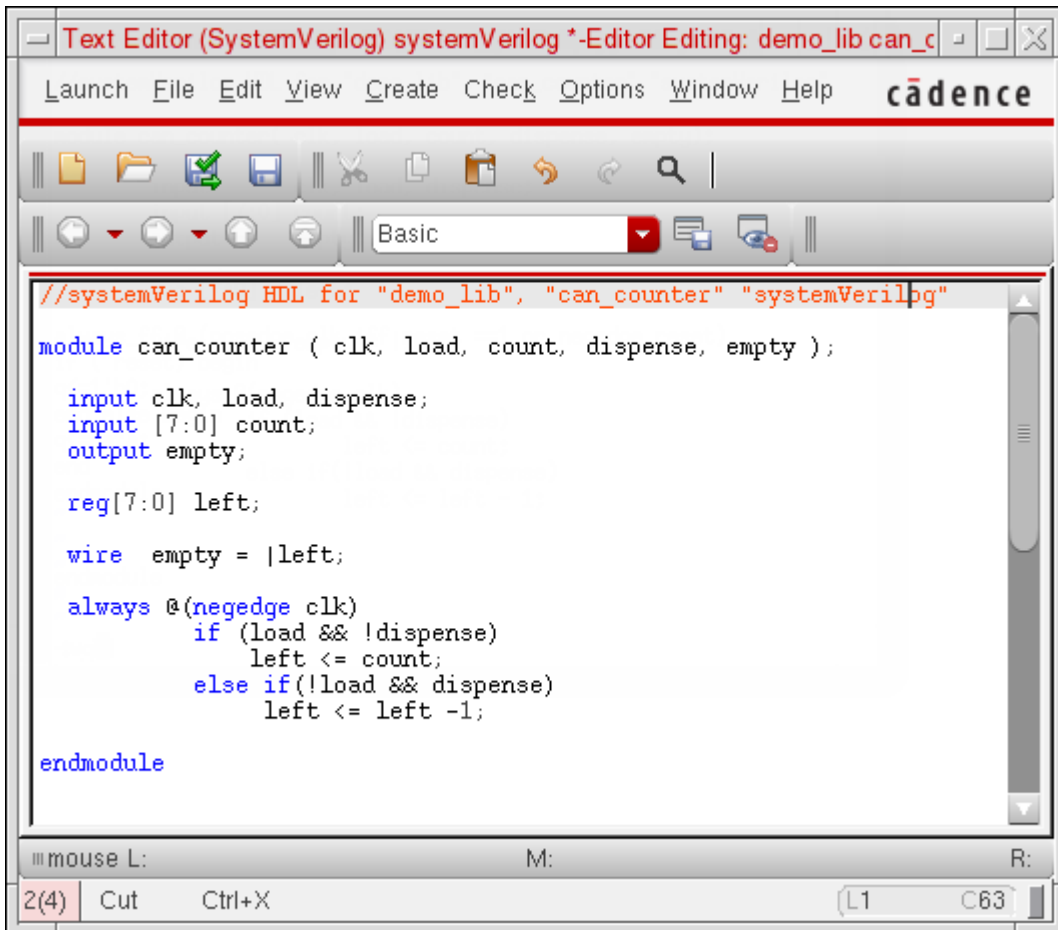
3. Click *OK*.

Virtuoso creates a blank SystemVerilog cellview and stores it as *LibraryName/cellName/viewName/verilog.sv*. It also displays this file in Virtuoso Text Editor.

4. Add your SystemVerilog cellview code and save the file.

Virtuoso parses the file. It prompts you to correct any syntax errors in the file.

The following figure illustrates an example of a SystemVerilog cell.



After parsing the file that does not have any syntax errors, Virtuoso prompts you to save its symbol.

5. Click Yes on the prompt to save the symbol of the new cell.

Note: Virtuoso CIW displays logs to indicate the status of the SystemVerilog cellview and symbol creation. Following are example logs:

```
Cellview can_counter symbol does not exist.
Symbol (can_counter symbol) generated and saved in library:demo_lib.
Processing Completed
    errors:0, warnings:0
```


Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Introducing SystemVerilog Integration Environment

To open the new cellview or symbol for verification:

1. Choose *File — Open* from the Virtuoso menu. The Open File form appears.
2. Select the library, cell, and view of the new cellview file. You can choose to view the symbol or the SystemVerilog text-only view.
3. Click *OK*. You can verify the text-only SystemVerilog view or symbol. See the following figure.

To create a SystemVerilog package view:

1. Choose *File — New — Cellview* from the Virtuoso menu. The New File form appears.



2. Specify the package view details.

Field	Details
<i>Library</i>	Select the library to which you want to add the new package.
<i>Cell</i>	Enter the name of the new package.
<i>View</i>	Specify the default view as required. If you leave this field blank and select a view type from the <i>Type</i> list, the field automatically displays the default view of the selected type,

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Introducing SystemVerilog Integration Environment

<i>Type</i>	Select <code>systemVerilogPackageText</code> from the list of view types. If the <i>View</i> field is blank, it updates to display <code>systemVerilogPackage</code> .
<i>Open with</i>	Select the default Text Editor.

3. Click *OK*.

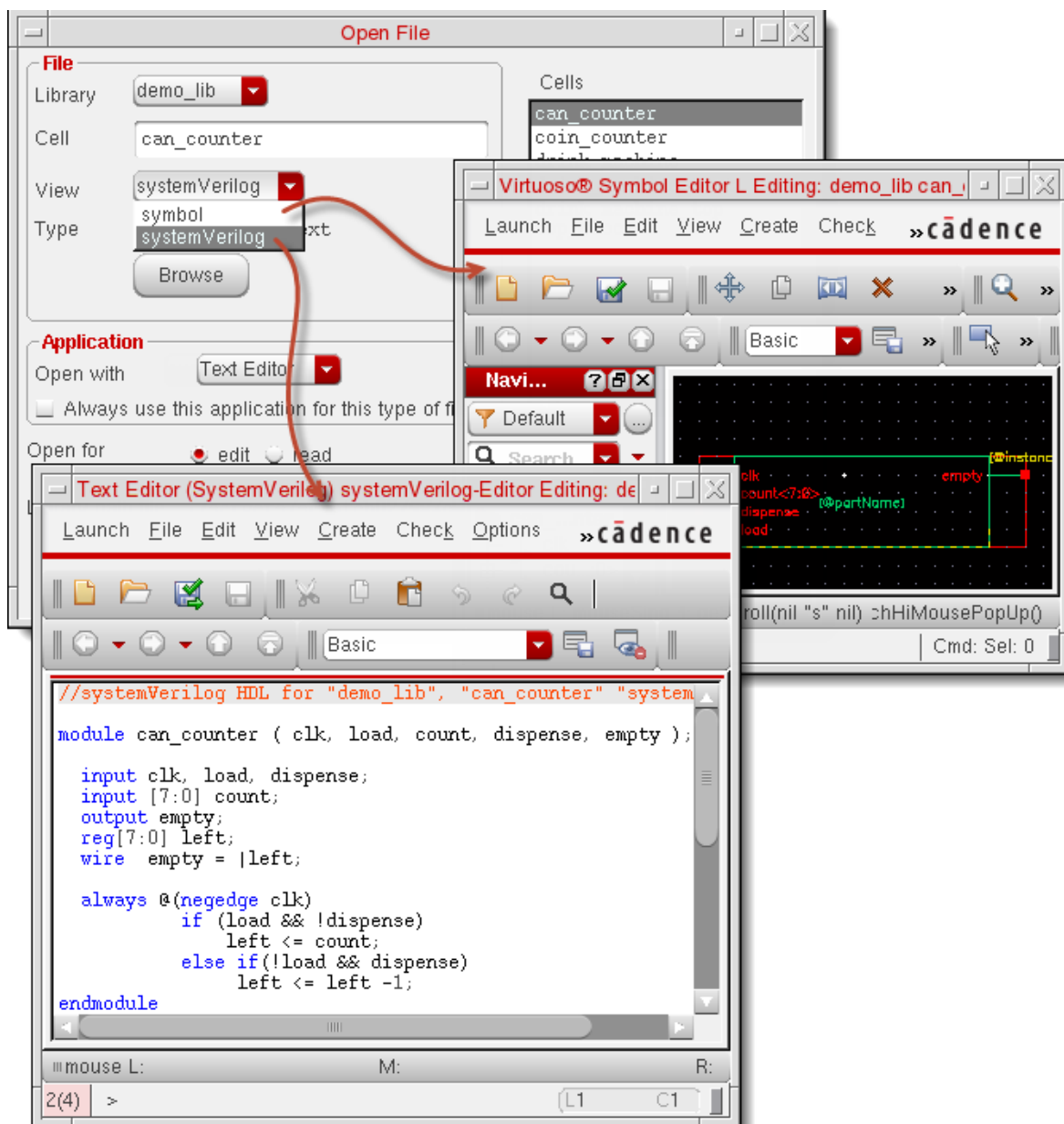
Virtuoso creates a blank SystemVerilog cellview and stores it as *LibraryName/cellName/viewName/package.sv*. It also displays this file in Virtuoso Text Editor.

4. Add your SystemVerilog package view code and close the file.

Virtuoso parses the file. It prompts you to correct any syntax errors in the file.

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Introducing SystemVerilog Integration Environment



Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Introducing SystemVerilog Integration Environment

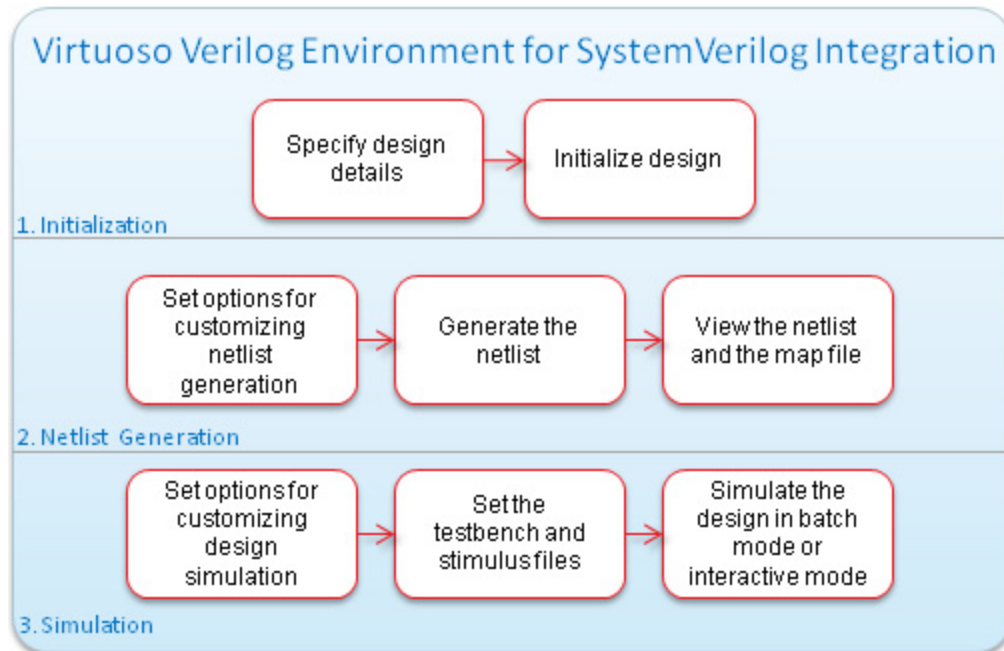
Generating Netlist and Simulating Designs

This chapter includes the following topics.

- [Overview](#) on page 22
- [Initializing the Run Directory](#) on page 23
- [Netlisting a Design](#) on page 26
- [Simulating a Design](#) on page 44
- [Using Standalone Mode](#) on page 57

Overview

The following figure shows how to use SystemVerilog Integration Environment for netlisting and simulating a design. For details on the functionality, features, generic flow, tool requirements, and the graphical user interface of this environment, see [Chapter 1, “Introducing SystemVerilog Integration Environment.”](#)



This chapter uses the following design example to illustrate the netlist generation and design simulation tasks.

Design Example

This guide uses a SystemVerilog-based drink machine design as an example to illustrate tasks performed using SystemVerilog Integration Environment. This design has the following SystemVerilog modules:

- `drink_machine` counts the amount of change that a user enters, dispenses a drink, and returns any due change.
- `coin_counter` loads the machine with coins and determines when the machine is out of coins.
- `can_counter` loads the machine with drinks and determines when the machine is empty.

The section [“About Creating SystemVerilog-Based Designs”](#) on page 14 illustrates the creation of the `can_counter` cellview and symbol.

The drink machine design kit includes the `ifc.v` file that contains the definition of some of the interfaces used in the design. This file must be included before the module declaration in the netlist. The drink machine design kit also includes `test_drink.v` that you can use as a testbench for initializing the machine and buying drinks.

This chapter assumes that you have launched the SystemVerilog Integration Environment for this drink machine design. For details, see [“Launching the Graphical User Interface”](#) on page 11.

Initializing the Run Directory

When a simulation is run on the Cadence system, all inputs and outputs of the simulation process are contained in a single directory. This directory is referred to as the run directory.

Initializing a run directory for a design means setting the environment for netlist generation and design simulation. When you initialize the run directory, the SystemVerilog Integration Environment adds some files and a directory to the run directory. It adds the `si.env` file to store netlist generation and design simulation settings. It also adds `.vlogifrc` to store Verilog and SystemVerilog format-specific configuration.

Note the following before initializing a run directory:

- If the run directory exists and contains the `.simrc` configuration file, the SystemVerilog Integration Environment loads settings from this file.

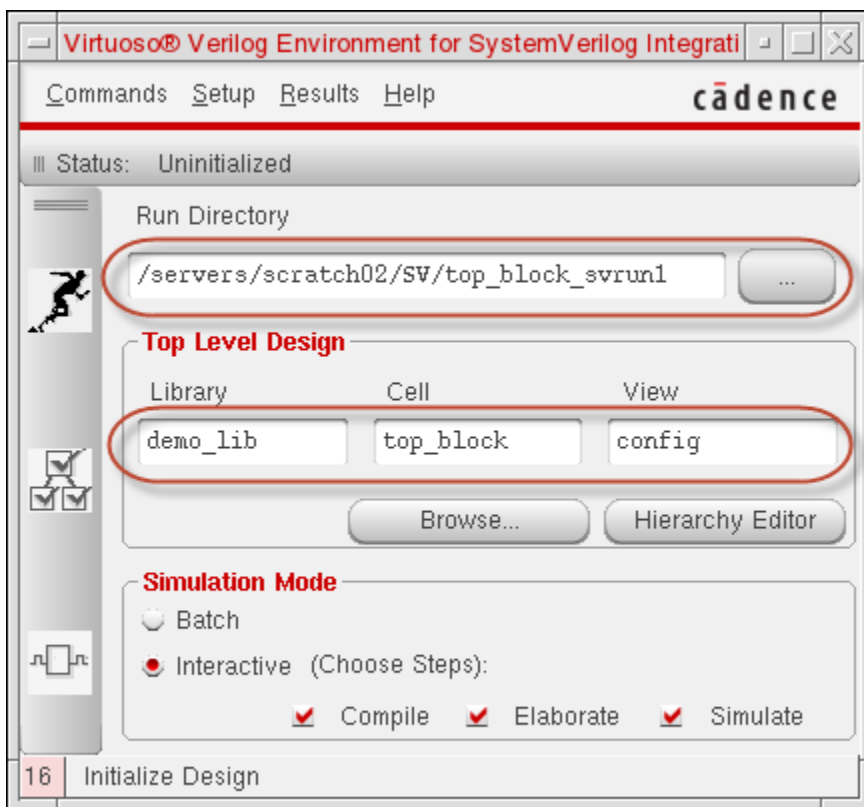
Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

- If the run directory exists and contains the `.vlogifrc` configuration file containing Verilog and SystemVerilog format-specific configuration flags, the environment loads settings from this configuration file.
- If the specified run directory does not exist, the environment loads the `$HOME/.vlogifrc` file. If the home directory does not contain `.vlogifrc`, the environment loads the default settings.
- If you are using a run directory and attempt to change it, the environment prompts you for confirmation whether you want to use the previous run directory settings for the new run directory.
- For details on `si.env` and `.simrc`, see the [Open Simulation System Reference](#).

To initialize the run directory for a design:

1. Ensure that the top-level design is correctly specified in the *Library*, *Cell*, and *View* fields of the main form.



By default, the SystemVerilog Integration Environment displays the library, cell, and view of the design opened in Virtuoso Schematic Editor, from where you launched the environment. To change the design, do one of the following:

- ☐ Click *Browse* and select the library, cell, and view using Library Browser.
- ☐ Enter the library, cell, and view in their respective fields.

If you choose the top-level configuration view of the design, you can click *Hierarchy Editor* and use Virtuoso® Hierarchy Editor to browse the design hierarchy and edit design configurations.

2. Specify the run directory.

By default, the SystemVerilog Integration Environment displays the directory *workingDirectory/topCellName_svrn1* in the *Run Directory* field. To change the run directory, do one of the following:

- ☐ Click the browse button and select the directory.
- ☐ Enter the path of the run directory.

You can specify a directory relative to the current working directory. The environment automatically expands the relative directory to its full path.

3. To initialize the run directory, do one of the following:

- ☐ Click *Initialize Design* on the toolbar.
- ☐ Choose *Commands — Initialize Design*.

The SystemVerilog Integration Environment does the following:

- ☐ Adds initial files and directory to the run directory
- ☐ Changes the *Status* on the main form from *Uninitialized* to *Ready*
- ☐ Enables the options to configure netlist generation and design simulation
- ☐ Enables the option to generate a netlist

Netlisting a Design

You can generate a netlist, which contains connectivity information of a design, after you have initialized a run directory for that design. Configure the netlist generation options before you generate the netlist. When you generate the netlist, the SystemVerilog Integration Environment creates a map file containing the netlist configuration options, and the map of the names used in the netlist and their corresponding names in the design. The environment lets you view the netlist and the map file.

This section provides information on the following topics.

- [Configuring Options for Generating a Netlist](#)
- [Managing Data Type Conflicts](#)
- [Overriding Hierarchical Data Type Propagation](#)
- [Ignoring Port Type Propagation](#)
- [Adding Port Properties to an Instance](#)
- [Generating a Netlist](#)
- [Viewing a Netlist and a Map File](#)

Configuring Options for Generating a Netlist

You can configure various options, based on which the SystemVerilog Integration Environment generates a netlist. The environment stores these configurations in the `si.env` and `.vlogifrc` files located in the run directory.

To configure netlist generation options after you have initialized the run directory:

1. Choose *Setup — Netlist*. The Netlist Setup form appears.
2. Set options as required.

For example, to configure netlist generation for the [sample drink machine design](#), you specify the path to `ifc.v` in the *Pre-Module Include File* field and retain other default settings. This include file contains the definition of interfaces. The SystemVerilog Integration Environment includes this file before the module declaration in the netlist.

3. Click *OK*.

Note: You can create a `.simrc` file in the run directory and store your default netlist generation and simulation configurations in this file. Configurations in `.simrc` overwrite configurations in `si.env`.

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

Note: By default, the environment uses the Verilog-2001 SystemVerilog ANSI format. To enable the Verilog 95 non-ANSI SystemVerilog format support, set `hnlPrintNonAnsiSV` to `t` in the `.simrc` file.

The following figure illustrates the Netlist Setup form and the table describes the configuration options. For information on the configuration flag associated with the fields in this form, see [“Configuring Flags for Netlist Generation”](#) on page 60.

Netlisting Mode: ☐ Entire Design ☒ Incremental ☐ Off

Netlist These Views: systemVerilog behavioral_sv functional_sv verilog_sv behavior

Netlist For LAI/LMSI Models: ☐

Generate SystemVerilog Test Fixture Template: ☒

Netlist Uppercase	<input type="checkbox"/>	Generate Pin Map	<input type="checkbox"/>	Preserve Buses	<input checked="" type="checkbox"/>
Netlist SwitchRC	<input type="checkbox"/>	Skip Null Port	<input type="checkbox"/>	Netlist Uselib	<input type="checkbox"/>
Drop Port Range	<input checked="" type="checkbox"/>	Incremental Config List	<input type="checkbox"/>	Symbol Implicit	<input type="checkbox"/>
Assign For Alias	<input type="checkbox"/>	Skip Timing Information	<input type="checkbox"/>	Declare Global Locally	<input type="checkbox"/>
Netlist Explicitly	<input type="checkbox"/>	Support Escape Names	<input type="checkbox"/>	Single Netlist File	<input type="checkbox"/>

Terminal SyncUp: Expand on Mismatch

Stop Netlisting at Views: systemVerilog verilog_sv 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

Pre-Module Include File: /servers/scratch02/SV/ifc.v

In-Module Include File:

OK Cancel Defaults Apply Help

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

Field	Description and Flag
<i>Netlisting Mode</i>	<p>Select the netlisting mode from:</p> <ul style="list-style-type: none">■ <i>Entire Design</i>: Netlists the entire design, regardless of which cells have been modified since the last netlist was generated.■ <i>Incremental</i>: Netlist only those parts of the design that have changed since the last netlist was generated.■ <i>Off</i>: Does not generate netlist of the design if another netlist exists. <p>Flag: <code>simReNetlistAll</code></p>
<i>Netlist These Views</i>	<p>Enter the views that you want to netlist for each cell or module. You use this option with the <u><i>Stop Netlisting at Views</i></u>. The environment 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.</p> <p>The default list is:</p> <pre>systemVerilog behavioral_sv functional_sv verilog_sv behavioral functional verilog schematic symbol</pre> <p>Flag: <code>verilogSimViewList</code></p>
<i>Netlist For LAI/ LMSI Models</i>	<p>Select if you want to use one of these simulation models:</p> <ul style="list-style-type: none">■ LOGIC Automation Incorporated (LAI) models: The <code>lai_verilog</code> property values indicate the use of these models■ Logic Modeling Systems Incorporated (LMSI) hardware simulation models: The <code>lmsi_verilog</code> property values indicate the use of these models. <p>Cells that do not have the <code>lai_verilog</code> or <code>lmsi_verilog</code> view type are netlisted according to the priorities established with the <u><i>Netlist These Views</i></u> and <u><i>Stop Netlisting at Views</i></u> options.</p> <p>Note: To attach the <code>lai_verilog</code> or <code>lmsi_verilog</code> view property to an instance on a Virtuoso Schematic Editor schematic, use the <i>Edit – Properties</i> command available on the schematic window. For details, see the <u><i>Virtuoso Schematic Editor L User Guide</i></u>.</p> <p>Flag: <code>simVerilogLaiLmsiNetlisting</code></p>

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

Field	Description and Flag
<i>Generate SystemVerilog Test Fixture Template</i>	<p>Select if you want the SystemVerilog Integration Environment to generate sample SystemVerilog testbench and stimulus files. You can edit these files and use them for simulating the design. For details, see “Specifying the Testbench File and Stimulus File” on page 51.</p> <p>Flag: <code>simVerilogTestFixtureFlag</code></p>
Netlist Control Options	<p>Select the following options to control netlist generation.</p> <ul style="list-style-type: none"> <p>■ <i>Netlist Uppercase</i>: Generates a netlist in uppercase letters.</p> <p>Flag: <code>vtoolsUseUpperCaseFlag</code></p> <p>■ <i>Generate Pin Map</i>: Creates the pin mapping files necessary to convert Standard Delay Files (SDF) pin names to SystemVerilog pin names. These files are stored in the <code>RunDirectory/pinMap</code> directory. Select this option only when you want to back annotate. Use this option when the pin names for a symbol in your schematic differ from those in the SystemVerilog library model description. After the pin map is created, the entire design is netlisted automatically to ensure that the netlister creates pin maps for the entire design.</p> <p>Note: The <i>Generate Pin Map</i> option must remain selected on subsequent runs. Otherwise, the netlister deletes your pin map directory.</p> <p>Flag: <code>hnlVerilogCreatePM</code></p> <p>■ <i>Preserve Buses</i>: Preserves buses (vectors) in the netlist. If this option is not selected, the netlister expands vector nets to single-bit equivalents (scalars) in the netlist.</p> <p>Flag: <code>simVerilogFlattenBuses</code></p> <p>■ <i>Netlist SwitchRC</i>: Includes user-defined RC switch properties in the netlist.</p> <p>Flag: <code>simVerilogHandleSwitchRCData</code></p> <p>■ <i>Skip Null Port</i>: Ignores floating instance ports.</p> <p>Flag: <code>simVerilogProcessNullPorts</code></p>

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

Field	Description and Flag
	<ul style="list-style-type: none">■ <i>Netlist Uselib</i>: Automatically adds the 'uselib directive to the netlist when a design includes two similarly named cells from two different libraries. Flag: <code>simVerilogHandleUseLib</code>■ <i>Drop Port Range</i>: Prints the module port without the port range. Flag: <code>simVerilogDropPortRange</code>■ <i>Incremental Config List</i>: Writes the renetlisted cellviews to the configuration list of the design. Flag: <code>simVerilogIncrementalNetlistConfigList</code>■ <i>Symbol Implicit</i>: Suppresses printing the net name during instance port formatting. Flag: <code>hnlVerilogNetlistStopCellImplicit</code>■ <i>Assign For Alias</i>: Uses an assignment statement for patches between nets. If you disable this option, the netlister applies the default <code>cds.alias</code> to patches between nets. Flag: <code>vlogifUseAssignsForAlias</code>■ <i>Skip Timing Information</i>: Ignore timing information assigned to instances in the design. Flag: <code>vlogifSkipTimingInfo</code>■ <i>Declare Global Locally</i>: Lets you declare global signals locally. When you disable this option, the netlister uses the default signals (<i>Global Power Nets</i> and <i>Global Ground Nets</i>). Flag: <code>vlogifDeclareGlobalNetLocal</code>

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

Field	Description and Flag
	<ul style="list-style-type: none">■ <i>Netlist Explicitly</i>: Generates name-based port lists using the connection-by-name syntax for modules in views. If you disable this option, the netlister generates 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 modules and order-based port lists are created for the instances of these modules. Flag: <code>simVerilogNetlistExplicit</code> Note: Even when you enable this option, instances of behavioral modules will still be connected implicitly. To get explicit connections for behavioral modules use the <code>hnlVerilogNetlistBehavioralExplicit</code> variable.■ <i>Support Escape Names</i>: Include escaped names in the netlist. It also allows you to escape names that are reserved keywords in SystemVerilog. Flag: <code>simVerilogEnableEscapeNameMapping</code>■ <i>Single Netlist File</i>: When selected, generates a single netlist instead of multiple netlists, one for each module. The netlist file is generated in the current simulation run directory. Flag: <code>simVerilogGenerateSingleNetlistFile</code>

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

Field	Description and Flag
<i>Terminal SyncUp</i>	<p>Specifies how to synchronize terminals between an instance and its switched master. You can choose from the following options:</p> <ul style="list-style-type: none">■ <i>Expand on Mismatch</i>: Retains the design terminals as is, unless there is a mismatch. In case of a mismatch, the netlister expands the mismatched terminals. Use this default option to generate a pure explicit netlist with flattened buses.■ <i>Honor Switch Master</i>: Always honors the switch master terminals.■ <i>Merge All</i>: Merges all terminals to create simple scalar and pure bus terminals. The resulting netlist does not have any bundles or split buses. <p>Note: You can also set the <code>hnlVerilogTermSyncUp</code> variable in the <code>.vlogifrc</code> file. The possible values are <code>mergeAll</code>, <code>honorSM</code>, and <code>nil</code> (default).</p> <p>Flag: <code>hnlVerilogTermSyncUp</code></p>
<i>Stop Netlisting at Views</i>	<p>Enter the list of views that 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 in this list, the netlister stops expansion of the design for this cell. The order of views in the stop list is irrelevant.</p> <p>The default list is:</p> <pre>systemVerilog verilog_sv verilog symbol</pre> <p>Flag: <code>verilogSimStopList</code></p>
<i>Global Power Nets</i>	<p>Enter 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.</p> <p>Flag: <code>simVerilogPwrNetList</code></p>
<i>Global Ground Nets</i>	<p>Enter 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.</p> <p>Flag: <code>simVerilogGndNetList</code></p>

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

Field	Description and Flag
<i>Global TimeScale Overwrite Schematic TimeScale</i>	<p>Select to overwrite any time values or units defined within a schematic with the following global variables:</p> <ul style="list-style-type: none"> ■ <i>Global Sim Time and Unit</i> Flags: <code>simVerilogSimTimeValue</code> and <code>simVerilogSimTimeUnit</code> ■ <i>Global Sim Precision and Unit</i> Flags: <code>simVerilogSimPrecisionValue</code> and <code>simVerilogSimPrecisionUnit</code> <p>The unit can be <i>s</i>, <i>ms</i>, <i>us</i>, <i>ns</i>, or <i>ps</i>.</p>
<i>Pre-Module Include File</i>	<p>Enter or select the file that the netlister must use as the include file before the module declaration in the netlist file generated for each hierarchical cellview.</p> <p>If <code>hnlVerilogDumpIncludeFilesInNetlist</code> is set to <code>t</code>, the content of the include file is copied to the netlist, instead of an <code>'include</code> statement.</p> <p>Flag: <code>vlogifPreModuleIncludeFile</code></p>
<i>In-module Include File</i>	<p>Enter or select the file that the netlister must use as an include file immediately after the module declaration in the netlist file generated for each hierarchical cellview.</p> <p>If <code>hnlVerilogDumpIncludeFilesInNetlist</code> is set to <code>t</code>, the content of the include file is copied to the netlist, instead of an <code>'include</code> statement.</p> <p>Flag: <code>vlogifInModuleIncludeFile</code></p>

Note: By default, the text files of *Stop Netlisting at Views* are included in the generated netlist. You can use `vlogifVicSVTextCellViewList` to include text files, in addition to the mandatory SystemVerilog text file. Use the following syntax in Virtuoso CIW or `si.env`:

```
vlogifVicSVTextCellViewList = (list "view_type_1" "view_type_2" "view_type_n")
```

For example:

```
vlogifVicSVTextCellViewList = (list "systemVerilogText" "text.v")
```

In this example, `systemVerilogText` represents the SystemVerilog view type and `text.v` represents the Verilog view type.

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 flags need to be set to true in the `.simrc` file or at CIW:

```
hnlVerilogPrintCDFParamExplicit = t
hnlVerilogPrintOverriddenCDFParamOnly = t
**Does not print the default value of parameter

hnlVerilogPrintParamExplicitNonStopping = t
** Prints Verilog/CDF parameters for non stopping cells.
```

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. These will be netlisted in inline explicit format at instance>
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.

Managing Data Type Conflicts

The SystemVerilog Integration Environment supports data type propagation from leaf-level SystemVerilog cellviews to the top-level schematic design hierarchy. During data type propagation, conflicts can occur in the following cases.

- *Conflict:* An instance with a SystemVerilog data type and another instance with no data type are connected to a pin.

Resolution: The SystemVerilog data type is propagated.

- *Conflict:* Two instances connected to a pin have different SystemVerilog data types.

Resolution: To determine which data type must be propagated, set a custom conflict resolution mechanism. For this, define the function

`hnlSVResolveDataTypeConflict` as required. The return value of this function must be `list(isRefPort portKind dataType isUnpacked)`.

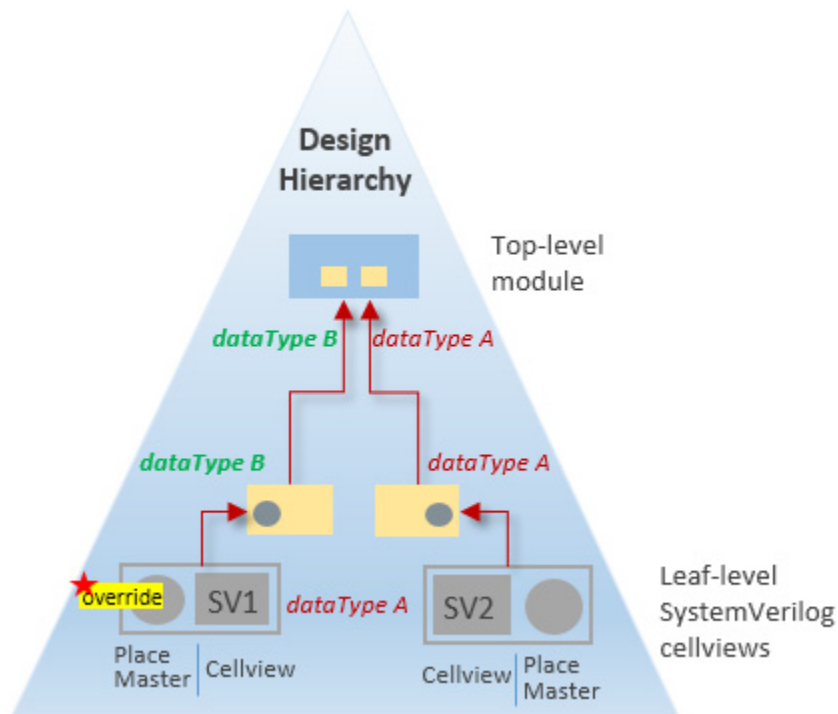
For further details and an example of resolving such a conflict, see [“Resolving Data Type Conflict”](#) on page 70.

Overriding Hierarchical Data Type Propagation

In a design hierarchy, SystemVerilog Integration Environment propagates data type from the leaf-level SystemVerilog cellviews to the top-level schematic. You can override the data type propagation from a port of a leaf-level SystemVerilog cellview to the top module in the hierarchy.

To override the data type propagation from a port, change the value of the `dataType` property of that port in the place master of the cellview to the data type you want to propagate. In this method, you do not need to change the data type of the port in the cellview.

The following figure illustrates the override. In this figure, SV1 and SV2 are SystemVerilog cellviews, each containing a port of data type A. In the place master of SV1, the data type of the port is overridden by data type B, which is further propagated in the design hierarchy.



For further details and an example of overriding data type propagation, see [“Overriding Hierarchical Data Type Propagation in a Design”](#) on page 68.

Note: If you do not want to override the data type propagation, set `vlogIfSVEnableDataTypeOverRiding` to `nil`.

Ignoring Port Type Propagation

If you have systemVerilog ports declared as `real var <port_name>` and generate a virtuoso schematic symbol, the real port will have `dataType = real` and `portKind = var`.

If you connect this port to another symbol port, which has `dataType = logic` and `portKind = wire`, you can successfully generate an NCVerilog /SystemVerilog netlist. However, if you try to use this netlist in an `irun` simulation, the elaboration fails due to incompatible port connection. To avoid this issue, you can ignore the data type and port information propagation by setting the `vlogIfSVDisableDataPropagation` flag to `t` in your `.simrc` file.

Adding Port Properties to an Instance

The Verilog Environment for SystemVerilog Integration allows you to modify the port properties `dataType` and `portKind`, which are specific to an instance. When `ignoreDataType` is set to `t`, the properties `dataType` and `portKind` are ignored. Instead, the `dataType` information that is propagated from the bottom-level cell to the top-level cell is considered.

You can add the `ignoreDataType` property on a specific instance terminal in the schematic. If this property is selected, the SystemVerilog Integration environment will not print the *Master Value* and the *Local Value*.

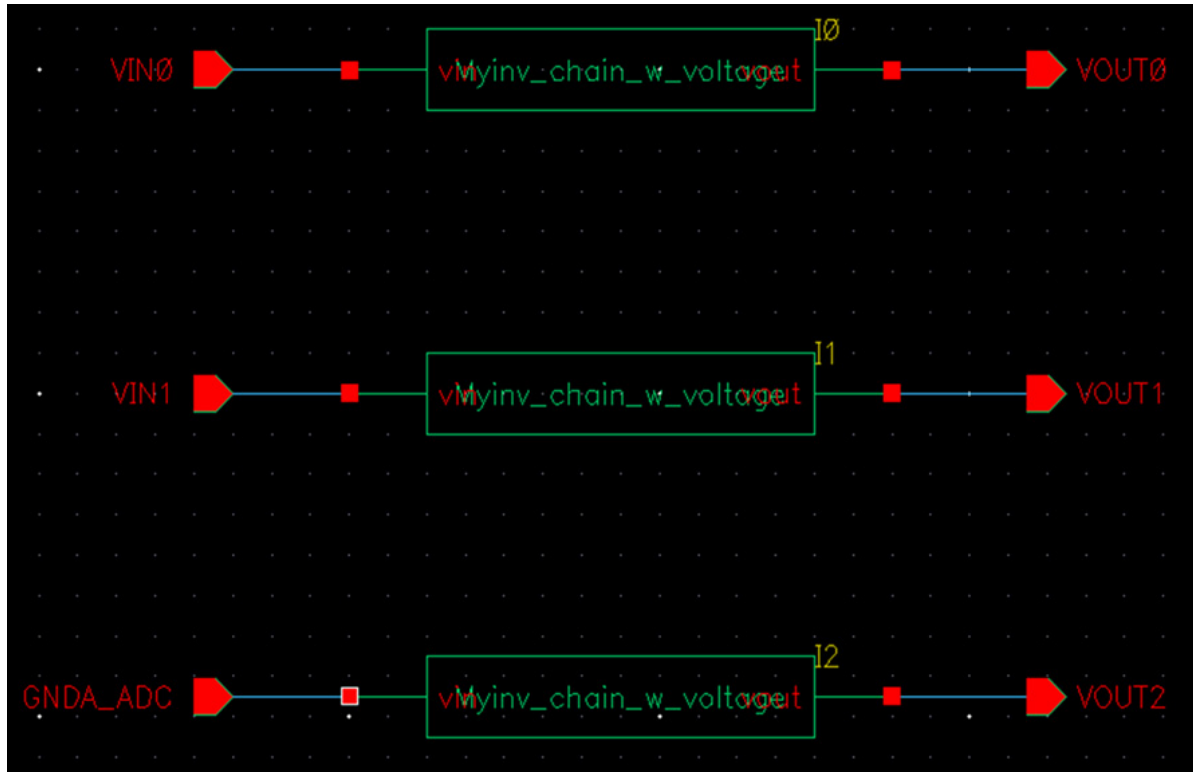
Additionally, you can modify the local values of the port properties `dataType` and `portKind` that are associated with a specific instance of a cell.

The following table clearly describes the impact of enabling and disabling the `ignoreDataType` property on the port of an instance in different scenarios:

Condition	Additional Condition	Result
<code>ignoreDataType = t</code>		The master values and local values of <code>dataType</code> and <code>portKind</code> are ignored. Instead, <code>dataType</code> information propagated from the bottom cell to the top cell is used.
<code>ignoreDataType = nil</code>	The local value of <code>dataType</code> is set to <i>custom_value</i> .	The local value of the specific instance is used instead of the <i>value</i> set on the symbol cell.
<code>ignoreDataType = nil</code>	The local values are not set for <code>dataType</code> and <code>portKind</code> .	The master value of the symbol cell property is used.

Example

Consider the input port I2 in the following schematic and the related condition scenarios that follow.



- When `ignoreDataType` is set to `t` is set on the port of an instance

When you set `ignoreDataType` to `t` on I2, the master and local values of `dataType` and `portKind` are ignored. In such a case, `dataType` information that is propagated from the bottom-level cell to the top-level cell is used.

Edit Object Properties

Apply To: only current instance pin

Show: ☒ user

Name: vin value

Direction: input

Signal Type: signal

Net Name: GNDA_ADC

Add Delete Modify

User Property	Master Value	Local Value	Display
dataType	voltage		off
isRefPort	false		off
portKind			off
ignoreDataType		<input checked="" type="checkbox"/>	off

OK Cancel Apply Defaults Previous Next Help

The following example shows how the netlist displays the port information in this scenario:

```
module tb_w_voltage (  
  output wire_logic VOUT0,  
  output wire logic VOUT1,  
  output wire logic VOUT2,  
  input wire logic GNDA_ADC,  
  input voltage VIN0,  
  input voltage VIN1 );
```

Here, the wire logic value is derived from the dataType property of the bottom-level cell.

- When `ignoreDataType` is set to `nil` and the local value of `dataType` is set to `custom_value`

When you set `ignoreDataType` to `nil` on I2 and the local value `dataType` to `myvoltage`, the local value `myvoltage` of the specific instance is used, instead of the master value `voltage` that is set on the symbol cell.

User Property	Master Value	Local Value	Display
dataType	voltage	myvoltage	off
isRefPort	false		off
portKind			off
ignoreDataType		nil	off

The following example shows how the netlist displays the port information in this scenario:

```
module tb_w_voltage (
output wire logic VOUT0,
output wire logic VOUT1,
output wire logic VOUT2,
input myvoltage GNDA_ADC,
input voltage VIN0,
input voltage VIN1 );
```

Here, the local value overrides the master value.

- When `ignoreDataType` is set to `nil` and the local value of `dataType` is not set

When you set `ignoreDataType` to `nil` on I2 and the local values of `dataType` and `portKind` are not set, the master value voltage of the symbol cell property is used.

Edit Object Properties

Apply To: only current instance pin

Show: ☒ user

Name: value

Direction:

Signal Type:

Net Name:

Add Delete Modify

User Property	Master Value	Local Value	Display
dataType	<input type="text" value="voltage"/>	<input type="text"/>	off
isRefPort	<input type="text" value="false"/>	<input type="text"/>	off
portKind	<input type="text"/>	<input type="text"/>	off
ignoreDataType	<input type="text"/>	<input type="checkbox"/>	off

OK Cancel Apply Defaults Previous Next Help

The following example shows how the netlist displays the port information in this scenario:

```
module tb_w_voltage (
output wire logic VOUT0,
output wire logic VOUT1,
output wire logic VOUT2,
input voltage GNDA_ADC,
input voltage VIN0,
input voltage VIN1 );
```

Here, the property of the symbol cell (master value) is used.

Generating a Netlist

You typically generate the hierarchical netlist of a design in the following cases:

- Before simulating a new design
- After modifying a cell in your design
- After modifying a cell instantiated from a source library

Before you generate the netlist of a design, ensure that the run directory is initialized and the netlist generation options are configured as required. SystemVerilog Integration Environment generates the netlist based on how you have configured these options. It also generates a map file that includes key netlist configurations and the map of the names used in the netlist and their corresponding names in the design.

Notes:

- SystemVerilog Integration Environment initializes all the OSS tabular name mapping variables to ensure that the generated netlist contains all the names in SystemVerilog namespace. It maps the OSS names for the design to SystemVerilog constructs, which is listed in the map.
- SystemVerilog Integration Environment supports the netlisting of designs with ports of type packed and unpacked arrays. For details, see [“Netlisting a Design Containing Packed and Unpacked Arrays”](#) on page 73.

SystemVerilog Integration Environment stores the `netlist` and `map` files in the `runDirectory/inhl/cds0` directory. It also creates various files and folders in the run directory.

To generate a netlist and a map file of a design based on netlist configuration, do one of the following:

- Click *Generate Netlist* on the toolbar.
- Choose *Commands — Generate Netlist*.

SystemVerilog Integration Environment does the following:

- ☐ Generates the netlist and map files and stores them in the run directory.
- ☐ Changes the *Status* on the main form from `Ready` to `Netlisting Succeeded`.
- ☐ Enables the options to view the netlist and map file.
- ☐ Enables the option to simulate the design.

Viewing a Netlist and a Map File

After generating a netlist, you can view the hierarchical netlist containing the connectivity information of the design. You can also view the map file containing the netlist configuration options and the map of the names used in the netlist and their corresponding name in the design.

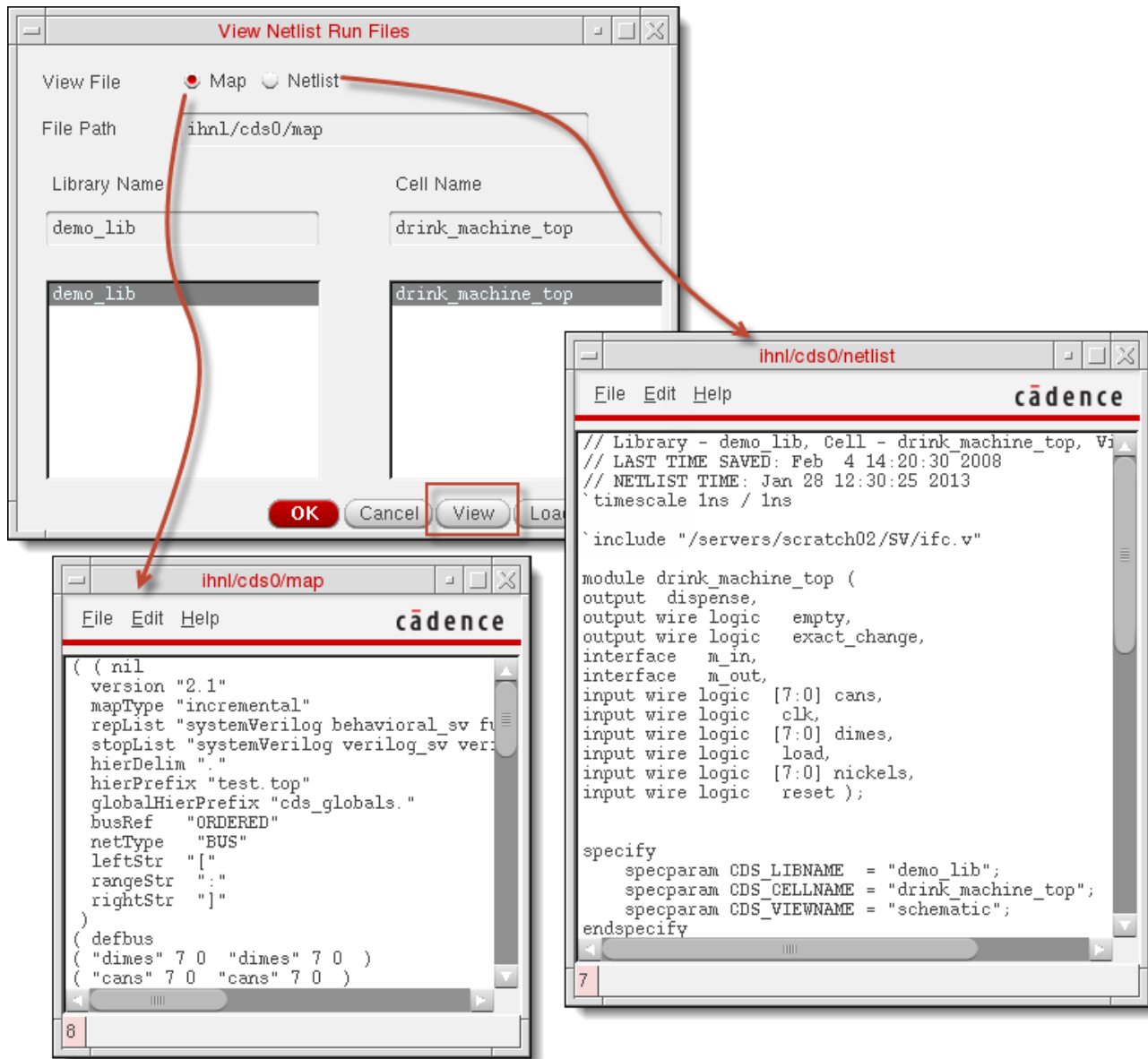
To view the netlist or the map file of a top-level design:

1. Choose *Results — Netlist*. The View Netlist Run Files form appears.
2. Select *Netlist* if you want to view the netlist content.
Select *Map* if you want to view the map file content.
3. Select the library from the *Library Name* list.
4. Select the cell from the *Cell Name* list. The *File Path* field displays the location of the netlist or map file.
Note: If you want to refresh the list of libraries and cells, click *Load*.
5. Click *View*. A text window appears with the file contents.

The following figure illustrates how SystemVerilog Integration Environment displays the netlist and the map file.

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs



Simulating a Design

Configure the design simulation options before you simulate a design for which you have generated a netlist. You can also specify the testbench and the stimulus you want to use for the simulation. You can then simulate the design in interactive or batch mode.

This section provides information on the following topics.

- [Configuring Options for Simulating a Design](#)
- [Specifying the Testbench File and Stimulus File](#)
- [Simulating a Design in Interactive Mode](#)
- [Simulating a Design in Batch Mode](#)

For information on using SystemVerilog Integration Environment in standalone mode, see [“Using Standalone Mode”](#) on page 57.

Configuring Options for Simulating a Design

You can configure various options, based on which SystemVerilog Integration Environment simulates a design. The environment lets you configure the simulation after you have initialized the run directory. It saves the Verilog and SystemVerilog format-specific settings in the `.vlogifrc` file. Settings in `.vlogifrc` overwrites the configurations in `si.env` and `.simrc`.

To configure simulation options:

1. Choose *Setup — Simulation*. The Simulation Setup form appears.
2. Set options as required.

For example, to configure the simulation of the [sample drink machine design](#), you configure the following fields as indicated and retain other default settings.

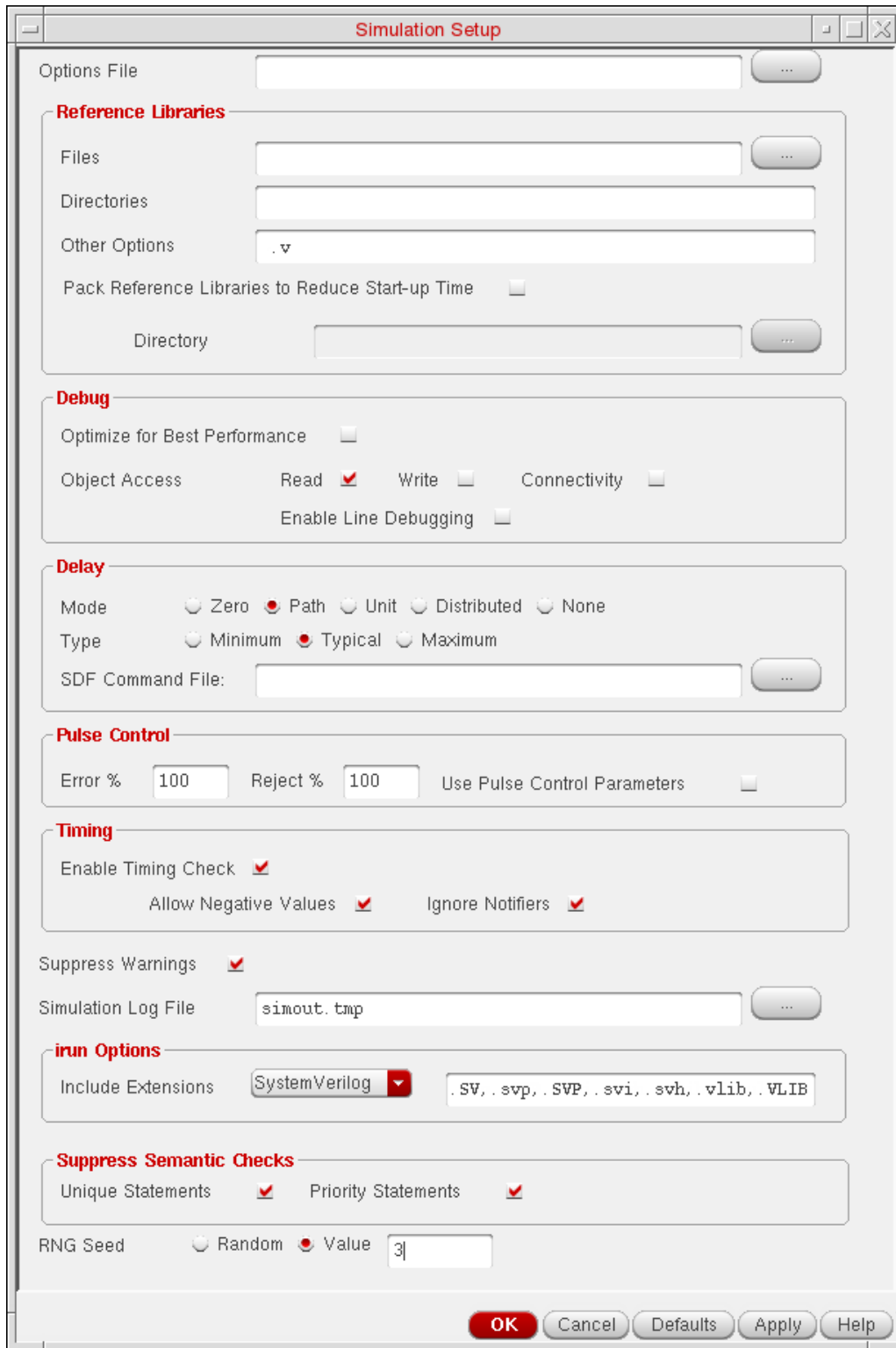
Field	Value
<i>Unique Statements</i>	Select
<i>Priority Statements</i>	Select
<i>RNG Seed > Value</i> checkbox	Select
<i>RNG Seed > Value</i> text field	3

3. Click *OK*.

The following figure illustrates the Simulation Setup form and the table describes the configuration options. For information on the configuration flag associated with the fields in this form, see [“Configuring Flags for Simulation”](#) on page 62.

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs



The **Simulation Setup** dialog box is used to configure simulation options. It contains several sections with various settings and checkboxes.

Options File: A text field for the options file, with a browse button (...).

Reference Libraries:
Files: A text field with a browse button (...).
Directories: A text field.
Other Options: A text field with a dropdown arrow.
Pack Reference Libraries to Reduce Start-up Time: ☐
Directory: A text field with a browse button (...).

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: A text field with a browse button (...).

Pulse Control:
Error %: 100 Reject %: 100 Use Pulse Control Parameters: ☐

Timing:
Enable Timing Check: ☒
Allow Negative Values: ☒ Ignore Notifiers: ☒

Suppress Warnings: ☒

Simulation Log File: A text field containing "simout.tmp" with a browse button (...).

Run Options:
Include Extensions: A dropdown menu showing "SystemVerilog" and a text field containing ".SV, .svp, .SVP, .svi, .svh, .vlib, .VLIB".

Suppress Semantic Checks:
Unique Statements: ☒ Priority Statements: ☒

RNG Seed: ☐ Random ☒ Value: 3

Buttons: OK, Cancel, Defaults, Apply, Help.

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

Field	Description and Flag
<i>Options File</i>	<p>Specify the path to the options file relative to the directory from which you started the Virtuoso session. The simulation is based on the SystemVerilog command options defined in the file. The file can also contain source text filenames or NC Verilog predefined + or - options that are not available through the interface.</p> <p>Flag: <code>simVerilogInvocationOptionsFile</code></p>
Reference Libraries	
<i>Files</i>	<p>Specify the path to the library files containing third-party SystemVerilog libraries used with this design. You can specify multiple paths.</p> <p>Flag: <code>simVerilogLibraryFile</code></p>
<i>Directories</i>	<p>Specify the path to the dedicated library directory containing third-party SystemVerilog library used with this design. You can specify multiple paths.</p> <p>Flag: <code>simVerilogLibraryDirectory</code></p>
<i>Other Options</i>	<p>Specify the file extensions that SystemVerilog Integration Environment must read in the specified directories.</p> <p>Flag: <code>simVerilogInvocationOptions</code></p>
<i>Pack Reference Libraries to Reduce Start-up Time</i>	<p>Select to specify a path where the compiler compiles the reference library files and directories.</p> <p>Flag: <code>simNCVerilogPackButton</code></p>
<i>Directory</i>	<p>Specify the path to the directory where the compiler must store the reference library files and directories.</p> <p>Flag: <code>simNCVerilogPackLib</code></p>
Debug	
<i>Optimize for Best Performance</i>	<p>Select to set the visibility access for all objects in the design. Use this option to increase the simulation performance by not giving read, write or connectivity access to the simulation objects.</p>

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

Field	Description and Flag
<i>Object Access</i>	<p>Select if you want to apply a specific type of access to the simulation object. The access can be:</p> <ul style="list-style-type: none">■ <i>Read</i>: Select to probe objects in the design and generate a Simulation History Manager (SHM) database or a Value Change Dump (VCD) database. This option lets you use SimVision to view waveforms, or SimCompare to compare the databases. Flag: <code>simNCVerilogReadAccess</code>■ <i>Write</i>: Select to specify values using the interactive simulation interface. For example, select this option when you use the <i>force</i> or <i>deposit</i> commands from the interactive simulation interface. Flag: <code>simNCVerilogWriteAccess</code>■ <i>Connectivity</i>: Select to display the load or driver information. For example, select this option if the driver command requires the load or driver information. Flag: <code>simNCVerilogConnectAccess</code> <p>The default value is <i>Read</i> access.</p>
<i>Enable Line Debugging</i>	<p>Select to use line break points and single stepping. It lets you break on a sequence execution and step in for debugging the design object.</p> <p>Flag: <code>simNCVerilogLineDebug</code></p>
Delay	

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

Field	Description and Flag
<i>Mode</i>	<p>Select the delay mode. You can choose from:</p> <ul style="list-style-type: none">■ <i>Zero</i>: Select to ignore all module path delays, timing checks, and structural and continuous assignment delays.■ <i>Path</i>: Select to use path delay information from specified blocks that contain module path delays; ignore structural and continuous assignment delays, with the exception of <code>triereg</code> charge decay times.■ <i>Unit</i>: Select to ignore module path delays and timing checks and convert all structural and continuous assignment delays that are nonzero to a single time unit.■ <i>Distributed</i>: Select to use the delay on nets, primitives, and continuous assignments and ignore module path delays.■ <i>None</i>: Select to use all of the delays in your netlist. <p>Flag: <code>simNCVerilogDelayMode</code></p>
<i>Type</i>	<p>Specify the delay type to apply during simulation. You can choose from:</p> <ul style="list-style-type: none">■ <i>Minimum</i>: Select to use all minimum delays.■ <i>Typical</i>: Select to use all typical delays.■ <i>Maximum</i>: Select to use all maximum delays. <p>Flag: <code>simNCVerilogDelayType</code></p>
<i>SDF Command File</i>	<p>Specify the name and path to the Standard Delay File (SDF) containing commands that the elaborator should annotate.</p> <p>If an SDF delay file already exists, the elaborator automatically generates an SDF command file and displays the filename in this field.</p> <p>Flag: <code>simNCVerilogSDFDFile</code></p>

Pulse Control

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

Field	Description and Flag
<i>Error</i>	<p>Specify the error limit percent. You typically set it to 0, 50, or 100 percent.</p> <p>Note: A pulse that is less than or equal to the specified percentage but greater than the reject limit, sets the module path output pulse to the <i>e</i> logic value.</p> <p>Flag: <code>simNCVerilogPulseCtlError</code></p>
<i>Reject</i>	<p>Specify the rejection percentage. You typically set it to 0, 50, or 100 percent.</p> <p>Flag: <code>simNCVerilogPulseCtlReject</code></p>
<i>Use Pulse Control Parameters</i>	<p>Select to read in the <code>PATHPULSE\$</code> special parameter from the SystemVerilog model in the SystemVerilog library, which overrides the global pulse control.</p> <p>Flag: <code>simNCVerilogPulseCtlSpecparam</code></p>
Timing	
<i>Enable Timing Check</i>	<p>Select to specify the following types of timing checks.</p> <ul style="list-style-type: none"> ■ Allow Negative Values: Select to allow negative values in <code>\$setuphold</code> and <code>\$recrem</code> timing checks in the SystemVerilog or Verilog description and in <code>SETUPHOLD</code> and <code>RECREM</code> timing checks in the Standard Delay Files (SDF) annotation. If this option is not set, negative values in the description or in the SDF annotation are set to 0 and a warning is generated. <p>Flag: <code>simNCVerilogTimingNeg</code></p> <ul style="list-style-type: none"> ■ Ignore Notifiers: Select to ignore notifiers in timing checks. <p>Flag: <code>simNCVerilogTimingNot</code></p> <p>Flag: <code>simNCVerilogEnableTimingCheck</code> for <i>Enable Timing Check</i></p>
<i>Suppress Warnings</i>	<p>Select to suppress all warnings during simulation.</p> <p>Flag: <code>simNCVerilogSupWarn</code></p>
<i>Simulation Log File</i>	<p>Specify the file to store simulation logs in the run directory. The default file is <code>simout.tmp</code>.</p> <p>Flag: <code>verilogLogFile</code></p>

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

Field	Description and Flag
irun Options	
<i>Include Extensions</i>	<p>Select the SystemVerilog or Verilog language whose file extensions you want to specify. In the text box, specify the comma-separated or space-separated list of file extensions.</p> <p>The default list for SystemVerilog is:</p> <pre>.template, .sv, .SV, .svp, .SVP, .svi, .svh, .vlib, .VLIB</pre> <p>The default list for Verilog is:</p> <pre>.v, .V, .vp, .VP, .vs, .VS</pre> <p>Note: The irun utility lets you simulate a mixed language design containing Verilog and SystemVerilog modules. During simulation, the irun utility processes files with the SystemVerilog and Verilog extensions specified in this field.</p>
Suppress Semantic Checks	
<i>Unique Statement</i>	<p>Select to suppress the semantic checking of unique constructs to shorten the simulation time.</p> <p>Flag: <code>simNCVerilogSVSuppressUnique</code></p>
<i>Priority Statements</i>	<p>Select to suppress the semantic checking of priority constructs to shorten the simulation time.</p> <p>Flag: <code>simNCVerilogSVSuppressPriority</code></p>
RNG Seed	
<i>Random</i>	<p>Select to randomly generate a seed value to initialize the random number generator (RNG) of the simulator.</p> <p>Flag: <code>simNCVerilogSVRNGSeed="random"</code></p>
<i>Value</i>	<p>Select to specify a seed value for the simulator and enter the value in the corresponding input field.</p> <p>Flag: <code>simNCVerilogSVRNGSeed="value"</code></p>

Specifying the Testbench File and Stimulus File

SystemVerilog Integration Environment lets you use the following files to initiate the simulation of a design:

- Testbench file containing instantiation of the complete design

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

- Stimulus file containing the preinitialized signal settings that drive a simulation

The testbench and stimulus files are referred to as the test fixture files.

You can set SystemVerilog Integration Environment to create the following sample test fixture files in the run directory during netlist generation. For this, select Generate SystemVerilog Test Fixture Template on the Netlist Setup form. The environment stores references to the test fixture files in the .vlogifrc configuration file.

- testfixture.template: The default testbench file
- testfixture.sv: The default stimulus file

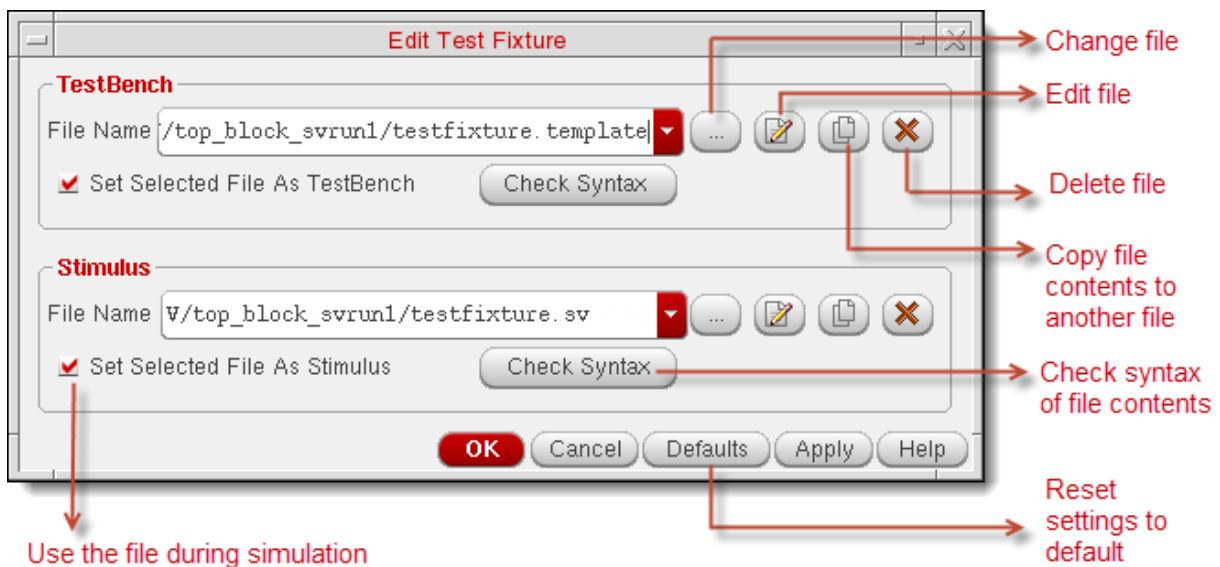
Important

The test fixture files should have the valid file extensions listed in the Include Extensions field on the Simulation Setup form. You must edit and check the contents of the sample test fixture files before you use them for simulation.

For information on the configuration flag associated with the fields in this form, see “Configuring Flags for Test Fixture Files” on page 63.

To work with the sample test fixture files:

1. Choose *Commands — Edit Test Fixture*. The Edit Test Fixture form appears.



Note: If you have generated a netlist for the first time in a new run directory, the names of the default files appear on the Edit Test Fixture form. Otherwise, the form displays the names of the files used during the previous simulation.

2. Edit and save the sample test fixture files as required.

Note: SystemVerilog Integration Environment generates the sample test fixture files with commented statements for reference, considering that a SystemVerilog module can have different types of ports at top-level. You must edit these files appropriately. Using the sample test fixture files without proper modification can causes the simulation to fail.

For the sample drink machine design, you can copy the contents of the `test_drink.v` file to the testbench file. The `test_drink.v` file is provided with the sample drink machine design kit. Also check the stimulus file contents.

3. Click *OK* to save the test fixture setting and close the form.

If you do not want to use the test fixture files for simulation, clear the *Generate SystemVerilog Test Fixture Template* checkbox on the Netlist Setup form and clear the *Set Selected File As TestBench/Stimulus* checkboxes on the Edit Test Fixture form. In this case, when you run the simulation in interactive mode, SystemVerilog Integration Environment launches SimVision without applying any stimulus. The batch mode does not require test fixture files.

Note: The flag for the testbench file name is `vlogifCurrentTestFixture`. The flag for the stimulus file name is `vlogifCurrentStimulus`.

Simulating a Design in Interactive Mode

Interactive mode of simulation lets you choose the compilation, elaboration, and simulation steps to perform. In this mode, SystemVerilog Integration Environment uses the NC tools and launches SimVision to let you interactively simulate the design. Using SimVision, you can directly interact with the ncsim simulator to open a database, trace signals, set breakpoints, observe signals, and perform other functions to verify your design.

Note: If you want to view simulation results in the SimVision window, set the SKILL variable `simNCVerilogNostdout` to `nil` in the Virtuoso CIW or in the simulation run control file `.simrc` before simulating the design. The default value of this variable is `t`.

To simulate a design in interactive mode:

1. Click *Interactive* in the *Simulation Mode* area of SystemVerilog Integration Environment main form.



2. Choose one of the following options.

- ☐ *Compile*
- ☐ *Compile and Elaborate*
- ☐ *Elaborate and Simulate*
- ☐ *Simulate*

Note: Before you can simulate a design, you must first compile the design and then elaborate it.

3. Do one of the following:

- ☐ Click *Simulate* on the toolbar.
- ☐ Choose *Commands — Simulate*.

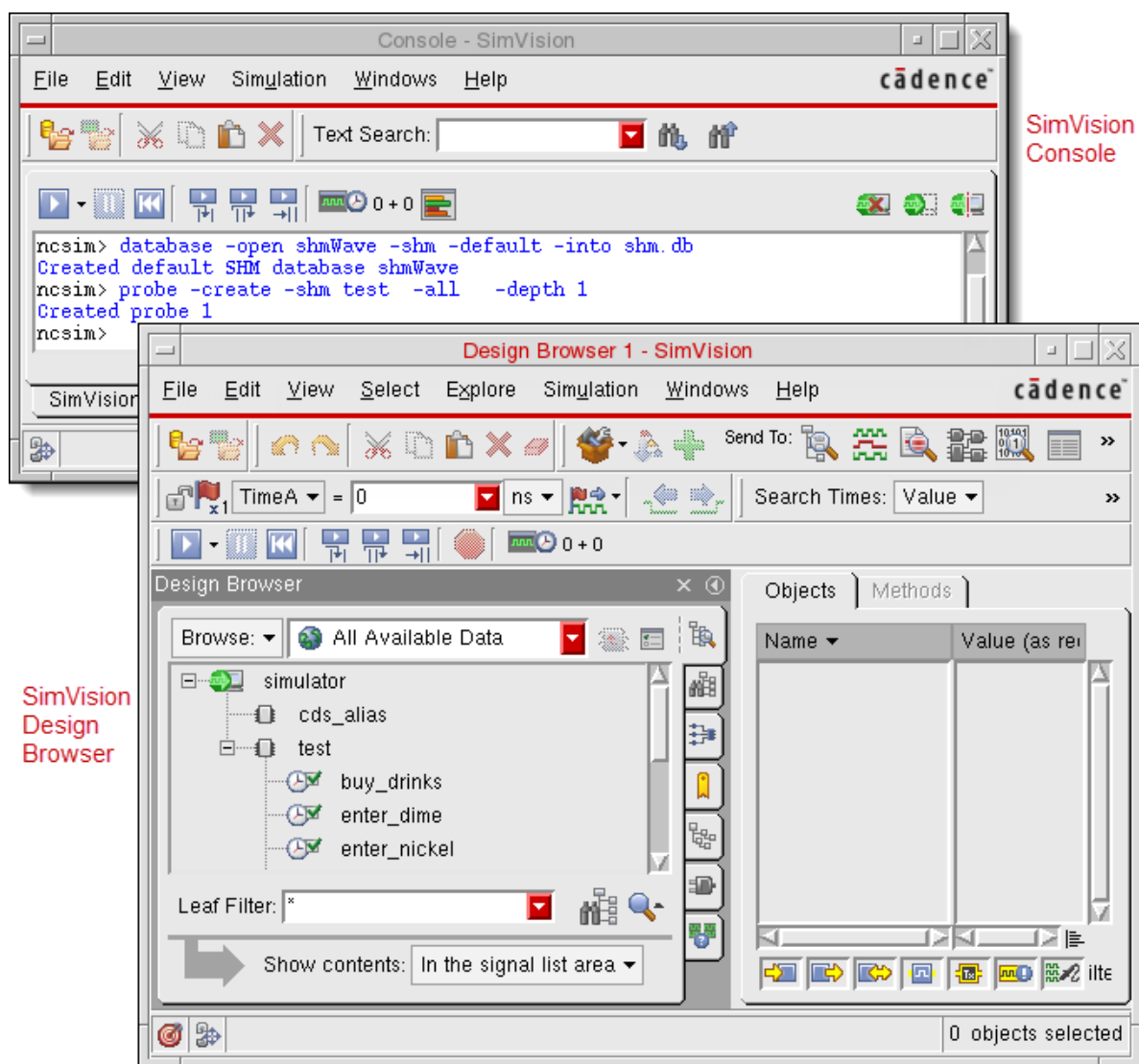
SystemVerilog Integration Environment performs the selected steps and informs you about the status.

If you chose the *Simulate* option, the environment launches SimVision. If you specified the test fixture files, the environment uses them to initiate the simulation. Otherwise, you must initiate the simulation.

The following figure illustrates SimVision, which has been launched to simulate the sample drink machine design. For details on using SimVision, see the *SimVision User Guide*.

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs



Simulating a Design in Batch Mode

Batch mode of simulation compiles, elaborates, and simulates the design in that order. It uses the NC tools for performing these steps. SystemVerilog Integration Environment provides a window to monitor the status of the batch simulation job.

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Generating Netlist and Simulating Designs

To simulate a design in batch mode:

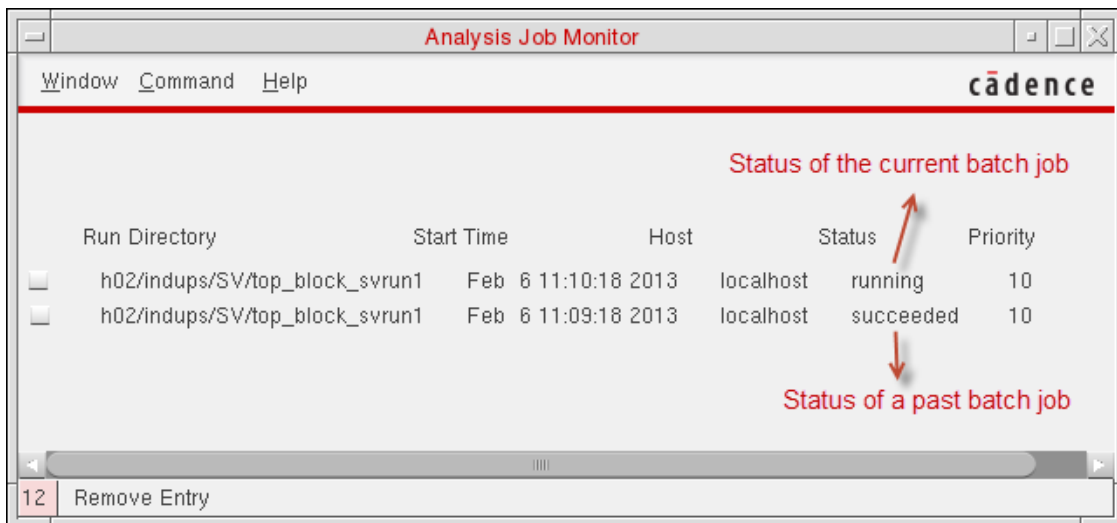
1. Click *Batch* in the *Simulation Mode* area of SystemVerilog Integration Environment main form.



2. Choose *Results — Job Monitor* to open the window for monitoring the batch simulation job. The Analysis Job Monitor window appears.
3. Do one of the following on SystemVerilog Integration Environment main form:
 - ❑ Click *Simulate* on the toolbar.
 - ❑ Choose *Commands — Simulate*.

SystemVerilog Integration Environment uses the NC tools to compile, elaborate, and simulate the design, without launching SimVision.

If the Analysis Job Monitor window is open, the environment displays the simulation status. This window also displays the status of the previous batch jobs. It includes an entry for each job and lets you prioritize, suspend, continue, or kill active jobs. For details on using this window, choose *Help* and see the *Simulation Environment Help*.



After completing the simulation, SystemVerilog Integration Environment displays a message, informing you about the simulation result.

Using Standalone Mode

You can run Open Simulation System-based SystemVerilog Integration Environment in standalone mode to generate a netlist of a top-level SystemVerilog-based design and simulate that design.

Cadence recommends that you initialize the run directory and configure the netlist and simulation options using the graphical user interface of SystemVerilog Integration Environment. If required, you can modify settings in the simulation environment configuration file `.simrc` or `si.env`. Note that settings in `.simrc` have precedence over settings in `si.env`.

To use standalone mode of SystemVerilog Integration Environment, ensure that the run directory includes `si.env` with the necessary configurations. If this file is absent, you cannot netlist and simulate a design in standalone mode.

Note: The `si.env` file is primarily used to instruct the simulation environment which design to simulate and which simulator to use. It is recommended not to include additional flags in this file because they can be lost after the simulation is run. Instead, use the `.simrc` file to customize simulation runs. Settings in this file overrides settings in `si.env`. For details, see *How SE Works in Open Simulation System Reference*.

You use the `si` command in the following syntax to generate a netlist and simulate a design.

```
si [run_directory] [-batch [-command commandName]] [-cdslib path]
```

In this syntax:

- `si` represents the Open Simulation System (OSS) binary file.
- `-batch` refers to the batch simulation mode.
- `-command` specifies the command to generate a netlist or simulate a design

You typically use the following `-command` options for your SystemVerilog-based designs:

- ☐ `netlist` instructs the simulation system to netlist the design.
- ☐ `vlogifNCSVSetupBatch` instructs the simulation system to simulate the design.
- `-cdslib` specifies the full path of `cds.lib`, which defines the design libraries and the path to these libraries.

For more information on the `si` command line interface, see the [*Open Simulation System Reference*](#).

The following examples illustrate how you use SystemVerilog Integration Environment in standalone mode.

- Run `si` from the run directory to netlist the design referred in `si.env`. The `cds.lib` file is located in the home directory.

```
si -batch -command netlist -cdslib ~/cds.lib
```

- Run `si` from the run directory to simulate a SystemVerilog-based design referred in the `si.env` file.

```
si -batch -command vlogifNCSVSetupBatch
```

Configuration Flags

This appendix lists the flags, or SKILL variables, associated with the configurable fields in SystemVerilog Integration Environment forms. In addition to setting these flags through the forms, you can set the flags in a configuration file, such as `si.env`, or from Virtuoso CIW. The appendix also lists additional useful flags that you can set in a configuration file or from Virtuoso CIW.

This appendix includes the following topics:

- [Configuration Flags for Design Details](#)
- [Configuring Flags for Netlist Generation](#)
- [Configuring Flags for Simulation](#)
- [Configuring Flags for Test Fixture Files](#)
- [Other Flags for Netlist Configuration](#)

Important Notes

- SystemVerilog Integration Environment is based on NC Verilog Environment. Therefore, they share some common configuration flags. Both the environments also share some features and capabilities.
- You can use various SKILL functions for formatting netlist output. These functions are specified as values for the `verilogFormatProc` property and are common for NC Verilog Environment and SystemVerilog Integration Environment.
- For details on the SKILL functions for formatting netlist output and common configuration flags, see the *[Virtuoso NC Verilog Environment User Guide](#)*. Also see the *[Digital Design Netlisting and Simulation SKILL Reference](#)*.

Configuration Flags for Design Details

The following table lists the fields in the main SystemVerilog Integration Environment form and their flags. For details, see [“Initializing the Run Directory”](#) on page 23.

Flag	Field
	<i>Top Level Design</i>
simLibName	<i>Library</i>
simCellName	<i>Cell</i>
simViewName	<i>View</i>

Configuring Flags for Netlist Generation

The following table lists the fields in the Netlist Setup form and their flags. For details, see [“Configuring Options for Generating a Netlist”](#) on page 26.

Flag	Field
simReNetlistAll	<u>Netlisting Mode</u>
verilogSimViewList	<u>Netlist These Views</u>
simVerilogLaiLmsiNetlisting	<u>Netlist For LAI/LMSI Models</u>
simVerilogTestFixtureFlag	<u>Generate SystemVerilog Test Fixture Template</u>
	<i>Netlist Control Options:</i>
vtoolsUseUpperCaseFlag	<u>Netlist Uppercase</u>
hnlVerilogCreatePM	<u>Generate Pin Map</u>
simVerilogFlattenBuses	<u>Preserve Buses</u>
simVerilogHandleSwitchRCData	<u>Netlist SwitchRC</u>
simVerilogProcessNullPorts	<u>Skip Null Port</u>
simVerilogHandleUseLib	<u>Netlist Uselib</u>
simVerilogDropPortRange	<u>Drop Port Range</u>
simVerilogIncrementalNetlistConfigList	<u>Incremental Config List</u>

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Configuration Flags

Flag	Field
hnlVerilogNetlistStopCellImplicit	<u>Symbol Implicit</u>
vlogifUseAssignsForAlias	<u>Assign For Alias</u>
vlogifSkipTimingInfo	<u>Skip Timing Information</u>
vlogifDeclareGlobalNetLocal	<u>Declare Global Locally</u>
simVerilogNetlistExplicit	<u>Netlist Explicitly</u>
	<p>Note: If this field is enabled or its flag is set to <code>t</code>, explicit netlisting is performed. If you run the netlister in standalone mode, you need to define this flag in the file <code>si.env</code>. Even when you use this option, instances of behavioral modules will still be connected implicitly. To get explicit connections for behavioral modules use the <u><code>hnlVerilogNetlistBehavioralExplicit</code></u> variable.</p>
simVerilogEnableEscapeNameMapping	<u>Support Escape Names</u>
simVerilogGenerateSingleNetlistFile	<u>Single Netlist File</u>
hnlVerilogTermSyncUp	<u>Terminal SyncUp</u>
verilogSimStopList	<u>Stop Netlisting at Views</u>
simVerilogPwrNetList	<u>Global Power Nets</u>
simVerilogGndNetList	<u>Global Ground Nets</u>
simVerilogOverWriteSchTimeScale	<u>Global TimeScale Overwrite Schematic TimeScale</u>
simVerilogSimTimeValue	<u>Global Sim Time</u>
simVerilogSimTimeUnit	<u>Global Sim Time Unit</u>
simVerilogSimPrecisionValue	<u>Global Sim Precision</u>
simVerilogSimPrecisionUnit	<u>Global Sim Precision Unit</u>
vlogifPreModuleIncludeFile	<u>Pre-Module Include File</u>
vlogifInModuleIncludeFile	<u>In-module Include File</u>

Configuring Flags for Simulation

The following table lists the fields in the Simulation Setup form and their flags. For details, see “Configuring Options for Simulating a Design” on page 45.

Flag	Field
simVerilogInvocationOptionsFile	Options File
simVerilogLibraryFile	Files
simVerilogLibraryDirectory	Directories
simVerilogInvocationOptions	Other Options
simNCVerilogPackButton	Pack Reference Libraries to Reduce Start-up Time
simNCVerilogPackLib	Directory
	Object Access
simNCVerilogReadAccess	Read
simNCVerilogWriteAccess	Write
simNCVerilogConnectAccess	Connectivity
simNCVerilogLineDebug	Enable Line Debugging
simNCVerilogDelayMode	Mode
simNCVerilogDelayType	Type
simNCVerilogSDFDFFile	SDF Command File
simNCVerilogPulseCtlError	Error
simNCVerilogPulseCtlReject	Reject
simNCVerilogPulseCtlSpecparam	Use Pulse Control Parameters
simNCVerilogEnableTimingCheck	Enable Timing Check
simNCVerilogTimingNeg	Allow Negative Values
simNCVerilogTimingNot	Ignore Notifiers
simNCVerilogSupWarn	Suppress Warnings
verilogLogFile	Simulation Log File
simNCVerilogSVSuppressUnique	Unique Statement

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Configuration Flags

Flag	Field
simNCVerilogSVSuppressPriority	Priority Statements
simNCVerilogSVRNGSeed="random"	Random
simNCVerilogSVRNGSeed="value"	Value

Example:

```
simNCVerilogSVRNGSeed="10"
```

Configuring Flags for Test Fixture Files

The following table lists the fields in the Edit Test Fixture form and their flags. For details, see [“Specifying the Testbench File and Stimulus File”](#) on page 51.

Flag	Field
vlogifCurrentTestFixture	TestBench File Name
vlogifCurrentStimulus	Stimulus File Name

Other Flags for Netlist Configuration

The following table describes some other flags for netlist configuration.

Flag	Field
vlogIfSVEnableDataTypeOverRiding	<p>Sets the data type propagation.</p> <p>By default, this flag is set to <code>t</code> to enable data type propagation. For details, see “Overriding Hierarchical Data Type Propagation” on page 35.</p> <p>For an example of data type propagation, see “Overriding Hierarchical Data Type Propagation in a Design” on page 68.</p>

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Configuration Flags

Flag	Field
<code>simSVPortPropertyList</code>	<p>When enabled, allows specifying only the <code>dataType</code> property and leaving the <code>portKind</code> property blank.</p> <p>Example:</p> <pre>simSVPortPropertyList = '(("analogLib" "res" "symbol" "vob_ana wrealldriver nil unPackedExplicit"))</pre> <p>This will print:</p> <pre>input wrealldriver vob_ana [1023:0];</pre>
<code>hnlPrintNonAnsiSV</code>	<p>Enables the Verilog 95 non-ANSI SystemVerilog format support.</p> <p>By default, SystemVerilog Integration Environment uses the Verilog-2001 SystemVerilog ANSI format. To enable the Verilog 95 non-ANSI SystemVerilog format support, set <code>hnlPrintNonAnsiSV</code> to <code>t</code> in the simulation run control file <code>.simrc</code>.</p> <p>For details on the <code>.simrc</code> file, see Customization of the Simulation Environment Using the .simrc File.</p>
<code>vlogPrintAssignForUnpacked</code>	<p>Enables the capability of printing the assign statements for the unpacked arrays of a design in the netlist.</p> <p>By default, this flag is set to <code>nil</code>. To print assign statements, set this flag to <code>t</code> in your netlist configuration file.</p> <p>For an example of a netlist with an assign statement, see Netlisting a Design Containing Packed and Unpacked Arrays.</p>
<code>vlogPrintConcatenationForUnpacked</code>	<p>Enables the printing of concatenation statements instead of assignment statements for unpacked arrays. For example, <code>{net1 net2}</code> instead of <code>'{net1 net2}</code>.</p> <p>By default, this flag is set to <code>nil</code>. To print the concatenation statements, set this flag to <code>t</code> in your netlist configuration file.</p>

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Configuration Flags

Flag	Field
vlogPrintAliases	Enables the printing of alias statements for signals connected to the patch element. For example, if signals connected to the patch element are <code>a</code> , <code>b</code> then, alias <code>a = b</code> will be printed.

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>`.

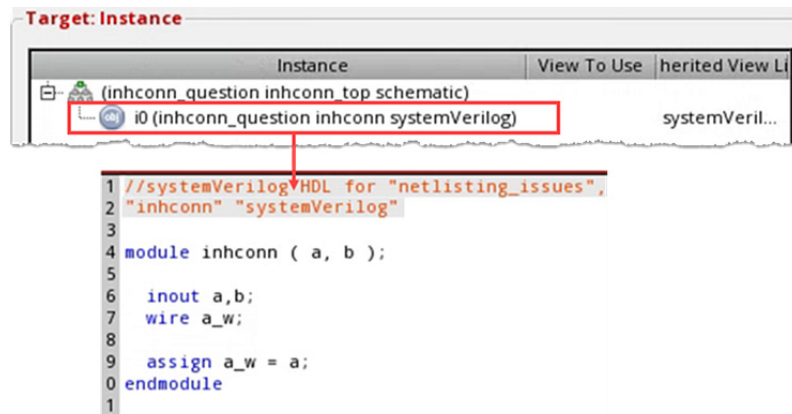
```
      a      z<1>      a      z<0>
---[src]~~>[dst]-----  ---[src]~~>[dst]-----
```

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`.

hnlUseSchematicForSystemVerilogView

A flag that supports printing inherited connections from the schematic when the instances are bound to a SystemVerilog view.

Consider an instance `i0` that is bound to a SystemVerilog view and has two ports `a`, `b` in the module definition.



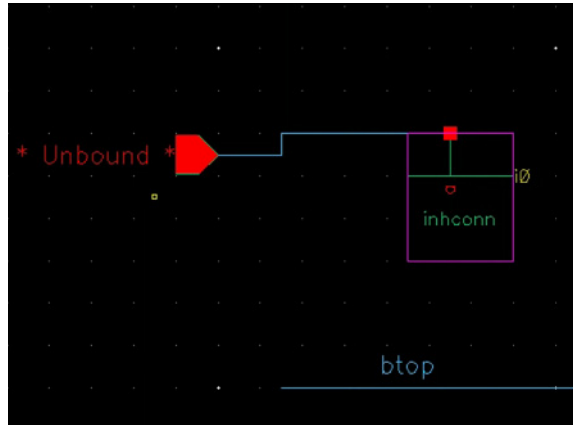
Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Configuration Flags

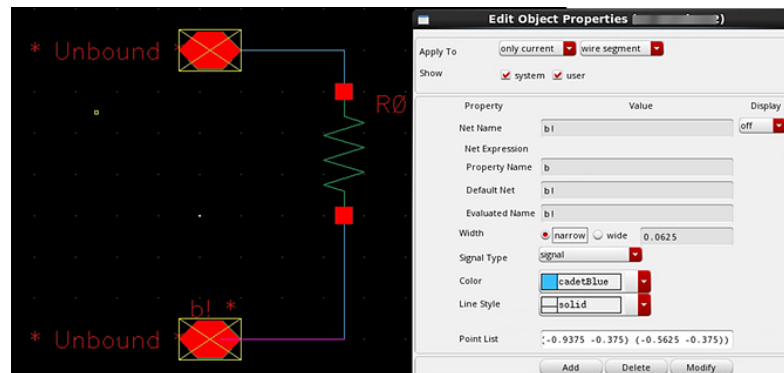
Flag

Field

In the top cell, the instantiation of `i0` has only one port, `a`.



However, an inherited connection is also created in the schematic view of `i0`.



When the instance is bound to a SystemVerilog view and you enable the `hnlUseSchematicForSystemVerilogView` flag, the netlist shows the inherited connections from the schematic view as shown below.

```
module inhconn top (
  input wire logic atop );
  wire btop ;

  specify
    specparam CDS_LIBNAME = "inhconn_question";
    specparam CDS_CELLNAME = "inhconn_top";
    specparam CDS_VIEWNAME = "schematic";
  endspecify

  inhconn i0 ( .a(atop), .b(btop));
endmodule
```

Examples

This appendix contains the following examples.

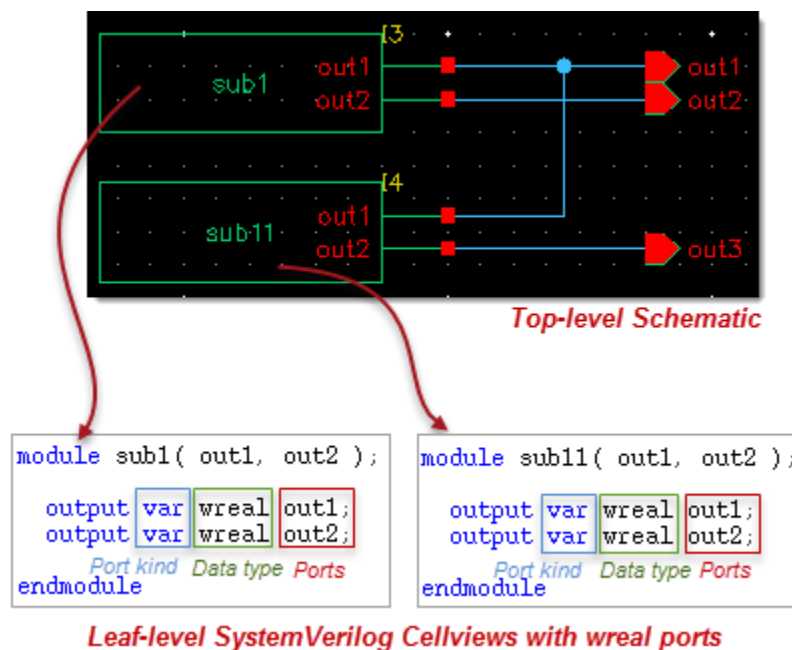
- [Overriding Hierarchical Data Type Propagation in a Design](#)
- [Resolving Data Type Conflict](#)
- [Netlisting a Design Containing Packed and Unpacked Arrays](#)

Overriding Hierarchical Data Type Propagation in a Design

You can override the data type propagation from a port of a SystemVerilog cellview in a hierarchical upwards fashion. For this override, change the value of the `dataType` property of that port in the place master, also referred to as the symbol, of the cellview to the data type you want to propagate. For details, see [“Overriding Hierarchical Data Type Propagation”](#) on page 35.

Note: If you do not want to override the data type propagation, set `vlogIfSVEnableDataTypeOverRiding` to `nil`.

As illustrated in the following figure, a top-level schematic contains instances of the SystemVerilog cellviews `sub1` and `sub11`. These cellviews have `wreal` ports.



When you generate the netlist of the top-level schematic, the `wreal` data type is propagated to the output ports of the schematic, which are `out1`, `out2`, and `out3`. The netlist is illustrated below. The bold text illustrates the `wreal` propagation.

```

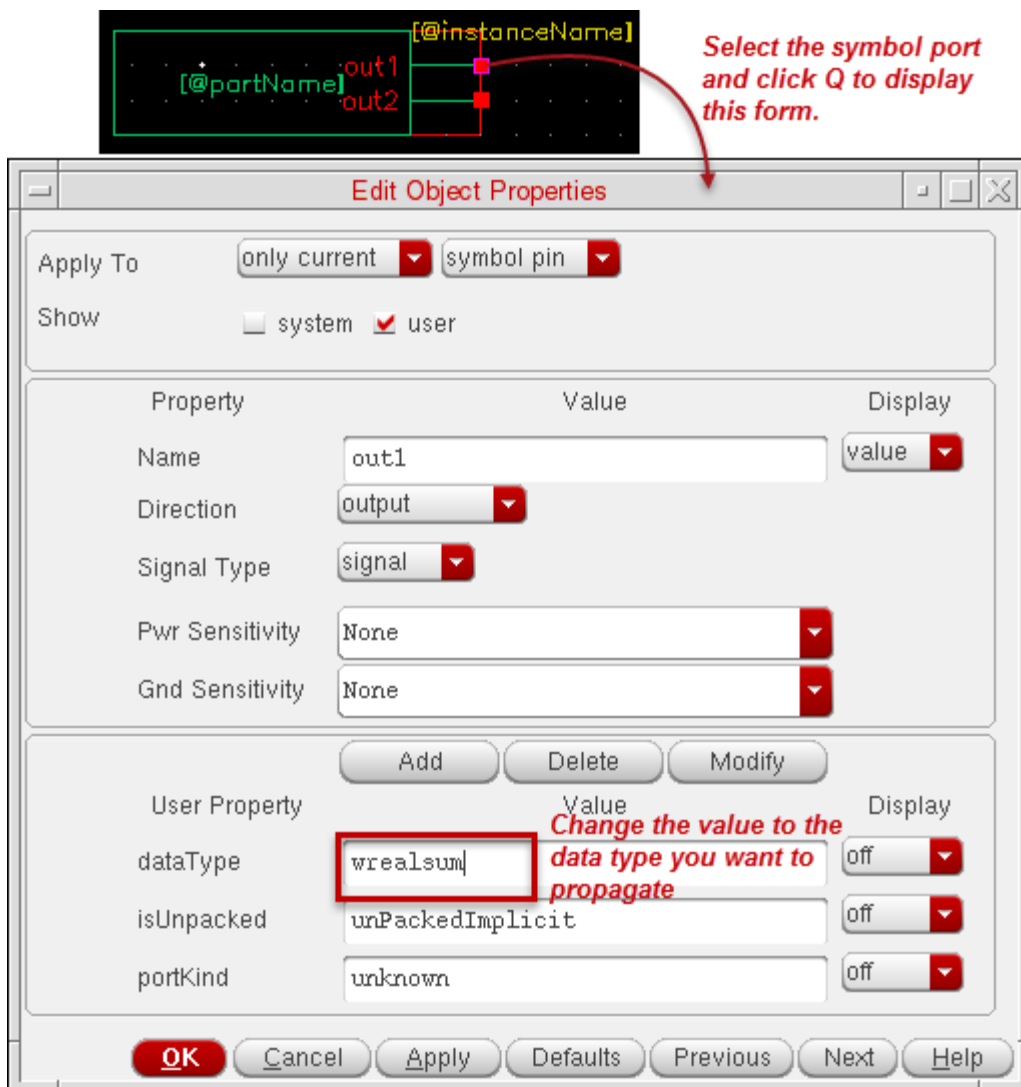
module top ( output wreal out1 , output wreal out2 , output wreal out3 );
sub1 I3 ( out1, out2);
sub11 I4 ( out1, out3);
endmodule
    
```

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Examples

You can override the `wreal` data type propagation with the `wrealsum` data type to add the signals `sub1.out1` and `sub11.out1` in the top module. To achieve this, perform the following steps:

1. Open the `sub1` symbol in Virtuoso Symbol Editor.
2. Select the port that represents `out1` and press Q to display the Edit Object Properties form of that port. See the following image.



3. Change the value of the `dataType` property from `wreal` to `wrealsum`.
4. Click **OK**.
5. Click *Check and Save* on the Virtuoso Symbol Editor toolbar. You can then close the editor.

6. Open the `sub11` symbol in Virtuoso Symbol Editor. Then perform [step 2](#) to [step 5](#) to change the `dataType` property value of the symbol port `out1` to `wrealsum`.

Note: At this point, you have overridden the data type in the symbols of the cellviews `sub1` and `sub11` to `wrealsum`. The cellviews still contain ports of data type `wreal`.

7. Open the top-level schematic that contains the updated symbols.
8. Generate the SystemVerilog netlist of the design. In this netlist, the data type `wreal` is overridden by `wrealsum`, as required.

The netlist where the data type is overridden is illustrated below. The bold text indicates the data type override.

```
module top ( output wrealsum out1 , output wreal out2 , output wreal out3 );  
sub1 I3 ( out1, out2);  
sub11 I4 ( out1, out3);  
endmodule
```

Resolving Data Type Conflict

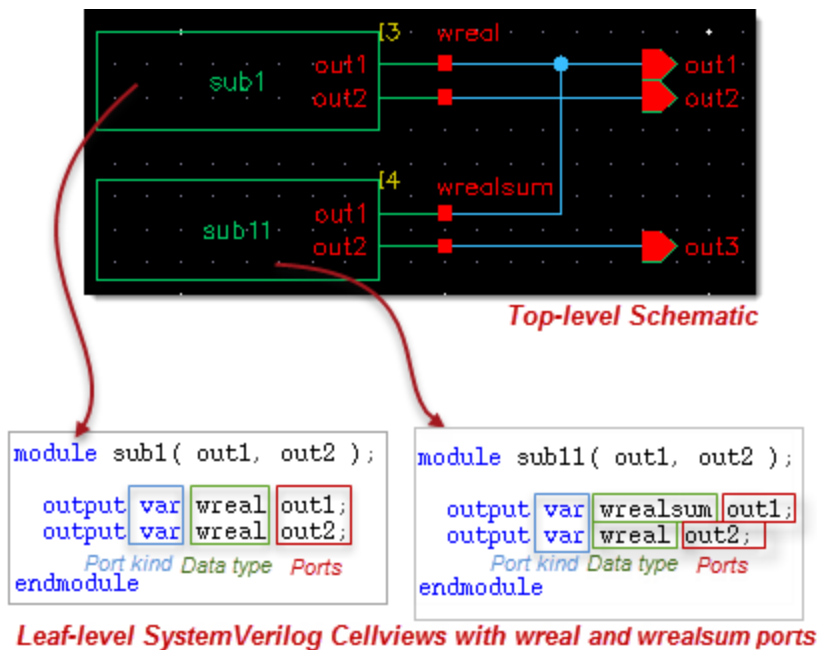
SystemVerilog Integration Environment can encounter conflicts when propagating data type. For details, see [“Managing Data Type Conflicts”](#) on page 35.

It is possible that two instances connected to a port have different SystemVerilog data types. For example, in the following design, the top-level schematic contains instances of the SystemVerilog cellviews `sub1` and `sub11`. Cellview `sub1` has `wreal` ports. Cellview `sub11` has port `out1` of the data type `wrealsum`, and port `out2` of the data type `wreal`.

Note: Port `out1 (wreal)` of `sub1` and port `out1 (wrealsum)` of `sub11` are connected to the same output port `out1`.

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Examples



When you generate the netlist of the illustrated design, SystemVerilog Integration Environment encounters a conflict when propagating the data type to the output port `out1` because the connected ports have different data types.

By default, the SystemVerilog Integration Environment does not propagate the data type to the output port in case of a type mismatch. However, you can set the `hn1SVSetDefaultTypeOnTypeConflict` flag to `t` to propagate the default data type when there is data type conflict.

Alternately, you can also set a custom conflict resolution mechanism to determine which data type must be propagated. For this, define the function `hn1SVResolveDataTypeConflict` as required and load it. For example, you can define and load the function through the simulation run control file `.simrc`. The return value of this function must be `list(isRefPort portKind dataType isUnpacked)`.

Virtuoso Verilog Environment for SystemVerilog Integration User Guide

Examples

The following example procedure stored in `.simrc` instructs SystemVerilog Integration Environment to propagate the data type `wrealsum` when a data type conflict is encountered.

```
procedure( hn1SVResolveDataTypeConflict(dt1 dt2)
    let( (isDT1RefObj isDT2RefObj dt1PortKind dt2PortKind dt1DataType
        dt2DataType isDt1Unpacked isDt2Unpacked resolvedDT )
; Define conflict resolution mechanism

; Check whether dt1 is a reference object
isDT1RefObj = car(dt1)
; Check whether dt2 is a reference object
isDT2RefObj = car(dt2)

; Find the portkind of dt1
dt1PortKind = cadr(dt1)
; Find the portkind of dt2
dt2PortKind = cadr(dt2)

; Find the dataType of dt1
dt1DataType = caddr(dt1)
; Find the dataType of dt2
dt2DataType = caddr(dt2)
resolvedDT = dt1DataType

;Check whether dt1 is unpacked
isDt1Unpacked = caddr(dt1)
;Check whether dt2 is unpacked
isDt2Unpacked = caddr(dt2)

; Check if there is conflict between "real" "wreal" or "wrealsum" datatype then
resolve datatype to "wrealsum"
when( and( member(dt1DataType list("real" "wreal" "wrealsum"))
member(dt2DataType list("real" "wreal" "wrealsum")))
    resolvedDT = "wrealsum")
    resolvedDT = list(isDT1RefObj dt1PortKind resolvedDT isDt1Unpacked)
        ; return resolved dataType
    )
)
```

When you generate the netlist of the design after setting the data type conflict mechanism, SystemVerilog Integration Environment propagates the required data type. It generates the

following netlist for the schematic example illustrated in this section. The bold text highlights the data type propagation as a result of the defined conflict resolution mechanism.

```
module top ( output wrealsum out1, output wreal out2 , output wreal out3 );  
sub1 I3 ( out1, out2);  
sub11 I4 ( out1, out3);  
endmodule
```

If the conflict resolution mechanism was not set, SystemVerilog Integration Environment would generate the following netlist, where no data type is propagated to the output port out1.

```
module top ( output out1, output wreal out2 , output wreal out3 );  
sub1 I3 ( out1, out2);  
sub11 I4 ( out1, out3);  
endmodule
```

Netlisting a Design Containing Packed and Unpacked Arrays

SystemVerilog Integration Environment supports the netlisting of designs with ports of the type packed array or unpacked array. Consider a design library `test` that contains the following SystemVerilog definition of module `packed_unpacked_data`.

```
//systemVerilog HDL for "test", "packed_unpacked_data" "systemVerilog"  
module packed_unpacked_data(packed_array, unpacked_array);  
    output [7:0] packed_array;  
    output real unpacked_array [7:0];  
endmodule
```

In this module, the port `packed_array` is declared as a packed array of default type `logic`. The port `unpacked_array` is declared as an unpacked array of type `real`.

The symbol of the SystemVerilog cellview `packed_unpacked_data` is instantiated as `I0` in the schematic `mid`, as illustrated below.



Virtuoso Verilog Environment for SystemVerilog Integration User Guide

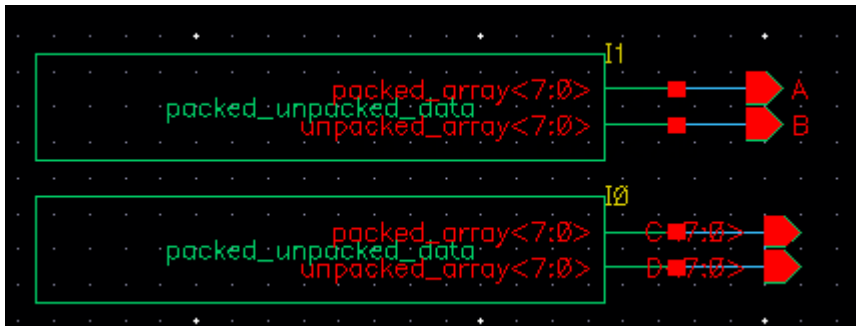
Examples

Note: Virtuoso provides support for handling packed and unpacked arrays during SystemVerilog symbol generation. For information on creating a SystemVerilog cellview and its symbol, see “[About Creating SystemVerilog-Based Designs](#)” on page 14.

When you generate the netlist of the schematic, SystemVerilog Integration Environment processes the design appropriately and generates the following netlist.

```
module mid ( input wire logic [7:0] C, input var real D [7:0] );
packed_unpacked_data I0 ( C[7:0], D[7:0]);
endmodule
```

Consider the following schematic `mid1`, where two instances of the symbol of `packed_unpacked_data` are instantiated as `I0` and `I1`. Note the pins A, B, C, and D.



When you generate the netlist of this schematic, SystemVerilog Integration Environment processes the design appropriately and generates the following netlist.

```
module mid1 ( output wire logic A, output var real B, input wire logic [7:0]
C, input var real D [7:0] );
packed_unpacked_data I1 ( {A, A, A, A, A, A, A, A}, '{B, B, B, B, B, B, B, B});
packed_unpacked_data I0 ( C[7:0], D[7:0]);
endmodule
```

If the flag `ylogPrintAssignForUnpacked` is set to `t` and SystemVerilog Integration Environment is restarted, the assign statements for the unpacked arrays are printed, as illustrated in the following netlist.

```
module mid1 ( output wire logic A, output var real B, input wire logic [7:0]
C, input var real D [7:0] );
var real cdsNet0[0:7];
assign cdsNet0 = {B, B, B, B, B, B, B, B};
packed_unpacked_data I1 ( {A, A, A, A, A, A, A, A}, cdsNet0);
packed_unpacked_data I0 ( C[7:0], D[7:0]);
endmodule
```

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	xmcls	xmcls.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