

# **Virtuoso SystemVerilog Netlister User Guide**

**Product Version ICADVM20.1  
October 2020**

© 2020 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.

**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

---

<u>Preface</u>	5
<u>Scope</u>	6
<u>Licensing Requirements</u>	6
<u>Related Documentation</u>	6
<u>What's New and KPNS</u>	6
<u>Installation, Environment, and Infrastructure</u>	6
<u>Virtuoso Tools</u>	7
<u>Other Tools</u>	7
<u>Additional Learning Resources</u>	7
<u>Video Library</u>	7
<u>Virtuoso Videos Book</u>	7
<u>Rapid Adoption Kits</u>	7
<u>Help and Support Facilities</u>	8
<u>Customer Support</u>	8
<u>Feedback about Documentation</u>	9
<u>Typographic and Syntax Conventions</u>	10
<b>1</b>	
<u>Introducing Virtuoso SystemVerilog Netlister</u>	11
<u>Additional Benefits of SystemVerilog Netlister</u>	13
<u>Prerequisites for Using SystemVerilog Netlister</u>	14
<u>Updating your .cshrc File</u>	15
<u>Netlist Generation Flow</u>	15
<u>Launching the SystemVerilog Netlister Interface</u>	16
<u>Understanding the Graphical User Interface</u>	18
<b>2</b>	
<u>Using SystemVerilog in Batch Mode</u>	21
<u>Creating Config Views of SystemVerilog Designs</u>	22
<u>Migrating SystemVerilog Integration Designs to SystemVerilog Designs</u>	22

## 3

### Managing Netlist Generation ..... 25

<u>Netlisting a Design</u> .....	26
<u>Specifying a Design</u> .....	26
<u>Setting Up Netlist Generation Options</u> .....	28
<u>Setting Up Design Variables</u> .....	38
<u>Generating a Netlist</u> .....	40
<u>Viewing a Netlist</u> .....	41
<u>Managing States</u> .....	41
<u>Saving States</u> .....	42
<u>Loading States</u> .....	43
<u>Customizing Netlist Generation Using .simrc</u> .....	44
<u>Adding Port Properties to an Instance</u> .....	46

### Environment Variables ..... 53

<u>isPortInANSIFormat</u> .....	54
<u>defaultNettype</u> .....	54
<u>enableDataPropagate</u> .....	55
<u>mergedNetlist</u> .....	55
<u>vlogSupply0Sigs</u> .....	56
<u>vlogSupply1Sigs</u> .....	56
<u>hdlVarFile</u> .....	57
<u>refLib</u> .....	57
<u>createXrunArgs</u> .....	58
<u>createXrunBinding</u> .....	58
<u>enableTimeScale</u> .....	59
<u>nettypesToIgnore</u> .....	59

### SKILL Functions ..... 61

<u>asiDigitalSimAutoloadProc</u> .....	61
--	----

---

# Preface

---

Virtuoso® SystemVerilog Netlister (SystemVerilog Netlister) is a utility for generating netlists of SystemVerilog digital designs.

This guide describes how to use the SystemVerilog Netlister to configure the environment for generating netlists of SystemVerilog designs.

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

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

This preface contains the following topics:

- [Scope](#)
- [Licensing Requirements](#)
- [Related Documentation](#)
- [Additional Learning Resources](#)
- [Customer Support](#)
- [Feedback about Documentation](#)
- [Typographic and Syntax Conventions](#)

## Scope

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

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

## Licensing Requirements

Virtuoso SystemVerilog Netlister requires the following licenses:

- Cadence Design Framework II license (License Number 111)
- Virtuoso Schematic Editor Verilog(R) Interface license (License Number 21400)
- Virtuoso AMS Designer Environment license (License Number 70000) or Virtuoso Schematic Editor XL license (License Number 95115)

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

## Related Documentation

### What's New and KPNS

- [\*Virtuoso SystemVerilog Netlister What's New\*](#)
- [\*Virtuoso SystemVerilog Netlister Known Problems and Solutions\*](#)

### Installation, Environment, and Infrastructure

- [\*Cadence Installation Guide\*](#)
- [\*Virtuoso Design Environment User Guide\*](#)

## Virtuoso Tools

### IC6.1.8 and ICADVM20.1

- [\*Virtuoso Schematic Editor User Guide\*](#)

## Other Tools

- [\*Xcelium Installation and Configuration\*](#)
- [\*Xcelium XRUN User Guide\*](#)

## Additional Learning Resources

### Video Library

The [Video Library](#) on the Cadence Online Support website provides a comprehensive list of videos on various Cadence products.

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

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

### Virtuoso Videos Book

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

### Rapid Adoption Kits

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

In addition, Cadence offers the following training courses on Virtuoso Space-based Router:

- Virtuoso Schematic Editor

Cadence also offers the following training courses on the SKILL programming language, which you can use to customize, extend, and automate your design environment:

- SKILL Language Programming Introduction
- SKILL Language Programming

To explore the full range of training courses provided by Cadence in your region, visit [Cadence Training](#) or write to [training\\_enroll@cadence.com](mailto:training_enroll@cadence.com).

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

## Help and Support Facilities

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

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

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

For more information, see [Getting Help](#) in *Virtuoso Design Environment User Guide*.

## Customer Support

For assistance with Cadence products:

- **Contact Cadence Customer Support**



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

■ Log on to Cadence Online Support

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

## Feedback about Documentation

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

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

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.
- On the Cadence Online Support [Product Manuals](#) page, select the required product and submit your feedback by using the *Provide Feedback* box.

## Typographic and Syntax Conventions

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

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

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

---

# Introducing Virtuoso SystemVerilog Netlister

---

Digital system verification uses the SystemVerilog language extensively, and this has introduced the Digital-Mixed-Signal (DMS) use model. The DMS use model allows discrete models to represent analog circuits. SystemVerilog allows you to use user-defined type and resolution functions, which make the net obsolete as a scalar object.

These use-model changes require a netlister that supports modern constructs, imports data from a design database, and produces a simulator-compatible netlist. A netlister that has these capabilities can traverse the design hierarchy to build the complete structure of a netlist. In such cases, the hierarchy can be a schematic view and a text view, or only a text view.

Virtuoso® SystemVerilog Netlister is a utility that helps you generate netlists of digital SystemVerilog designs. This utility imports configuration views of digital designs for netlist generation, directly parses and accesses SystemVerilog and Verilog text models and creates LRM-compliant SystemVerilog configurations to generate compatible netlists.

The following table lists the main features of SystemVerilog Netlister:

Main Features	Description
Dual Interface	Provides a graphical user interface and a command-line interface. Supports the batch or command-line mode using the <code>runsv</code> command.
LRM-compliant Netlist	Uses only SystemVerilog LRM constructs. This makes the netlist simulator-independent and available to a SystemVerilog-compliant tool. Vendor extensions are not supported.
HED Config	Fully supports HED configurations that match UNL. Netlist generation is based on HED configurations.

## Virtuoso SystemVerilog Netlister User Guide

### Introducing Virtuoso SystemVerilog Netlister

---

Main Features	Description
Direct Access to HDL File	<ul style="list-style-type: none"><li>■ Restricts the use of the direct access OA file.</li><li>■ Updating the OA for text or symbol for text-in-text instance is redundant.</li><li>■ Supports read-only libraries.</li><li>■ Supports handling ports of packed or unpacked arrays in the design.</li><li>■ Ensures accurate datatype propagation from leaf-level SystemVerilog and Verilog cellviews to top-level schematic views. The datatype propagation is based on the native datatype definitions from text files. By default, datatype propagation is disabled.</li><li>■ Ensures that instances are saved with explicit port connections. Supports smart connections with module ports of Verilog and SystemVerilog views.</li><li>■ Requires the following:<ul style="list-style-type: none"><li>□ Instance parameters are netlisted correctly and parameters are propagated in a UNL-supported method.</li><li>□ Instance parameters that differ from the default parameters use the explicit declaration format.</li></ul></li></ul>

---

Some of the advantages that SystemVerilog Netlister offers, which the NC Verilog Netlister does not, are as follows:

- Symbol is redundant for text-in-text instances
- OA update for text is redundant
- Schematic text sandwich structure
- Support for read-only libraries
- Full support for HED config
- Flexible flowchart control
- Advanced parameter and bus handling
- Simulator-independent netlist and binding

## Additional Benefits of SystemVerilog Netlister

SystemVerilog Netlister provides a quick and efficient approach to netlist SystemVerilog designs and provides the following additional features and capabilities:

Feature	Description
Schematic Text Sandwich Configuration	Supports the following formats: <ul style="list-style-type: none"><li>■ Text-in-schematic</li><li>■ Schematic-in-text</li><li>■ Text-in-text</li></ul>
Text on Top	Supports digital text-on-top views (Verilog, SystemVerilog). Allows descending into the leaf-level schematic or text views.
Creation of SV 2001 Config for Bindings	Supports: <ul style="list-style-type: none"><li>■ Same cell from different libraries</li><li>■ Multiple views of the same cell</li></ul>
Config in Config	Supports config-in-config views where config view has the top-level schematic of different cells.
Symbol Avoidance	Avoids the requirement for a symbol unless the cell is instantiated in a schematic view.
Data type Handling	Supports SystemVerilog data types. Requires that any net, port, or bus from a schematic or symbol uses the <code>interconnect</code> net type, by default. Support for the <code>wire</code> net type is also available. Allows declaring internal signals with these nettypes.
Instance Parameters Handling	Requires the following: <ul style="list-style-type: none"><li>■ Instance parameters are netlisted correctly and parameters are propagated in a UNL-supported method.</li><li>■ Instance parameters that differ from the default parameters use the explicit declaration format.</li></ul>
Port Connection Handling	Ensures that instances are saved with explicit port connections. Supports smart connections with module ports of Verilog and SystemVerilog views.

## Virtuoso SystemVerilog Netlister User Guide

### Introducing Virtuoso SystemVerilog Netlister

Feature	Description
Optional xrunArgs File Creation	Creates an <code>xrunArgs</code> file, which includes the full set of generated files.
Keywords Handling	Prefixes each 1800-2012 keyword with an escape character.
Inherited Connection Handling	Mandates the use of port-drilling only for schematic views.
Self-contained Netlist Creation	Creates a self-contained set of files without links to the text views in <code>dfl</code> .*.
ANSI Port Declaration	Supports module port declaration in ANSI format.
Iterated Instance Support	Supports instance arrays instead of flattened instances in the design.
Design Variable Support	Supports the use of design variables. All design variables are saved in the <code>cds_globals.sv</code> file.
Shorting Support	Supports shorting devices with two terminals.
Bind to Open Support	Supports binding instances to open ports.
Save/Load State	Supports saving states with specified settings and subsequently loading these saved states for reuse.
cdsenv Support	Provides various <code>.cdsenv</code> options for configuring netlist generation. Saves the values of these options into states.
CDF Parameter Support	Fully supports CDF parameters, including <code>pPar</code> .

## Prerequisites for Using SystemVerilog Netlister

Ensure that you have the following tools:

- Cadence Xcelium `xrun` utility 64-bit version (18.09 or higher)

The `xrun` utility helps you specify all input files and options in a single command.

It can take SystemVerilog designs as input. The utility uses the Cadence Native Code tools to compile and netlist designs.

- Cadence Virtuoso 64-bit version (IC6.1.8 ISR8 or higher)

SystemVerilog Netlister is launched from Virtuoso.

## Updating your .cshrc File

Ensure that you update your `.cshrc` file as follows:

- Specify the path to the Virtuoso installation.
- Specify the path to the Xcelium installation.



*Tip*

To ensure that you use the 64-bit version of Virtuoso and Xcelium, add the environment variable `setenv CDS_AUTO_64BIT ALL` to your `.cshrc` file.

## Netlist Generation Flow

The generic flow for using SystemVerilog Netlister is as follows:

1. Ensure that the environment is ready for using SystemVerilog Netlister. See [Prerequisites for Using SystemVerilog Netlister](#).
2. Launch SystemVerilog Netlister from Virtuoso. See [Launching the SystemVerilog Netlister Interface](#).
3. Specify your design. The design must have a SystemVerilog configuration (config) view at the leaf level. See [Specifying a Design](#).
4. Configure the options for generating a netlist of the specified design. See [Setting Up Netlist Generation Options](#).
5. Configure the design variables for the specified design. See [Setting Up Design Variables](#).
6. Generate a netlist of the design. See [Generating a Netlist](#).
7. View the netlist results as required. See [Viewing a Netlist](#).
8. Save settings as states or load saved states. See [Managing States](#).

To know about any issues that you might encounter while using SystemVerilog Netlister, refer to the log in the Virtuoso CIW. The status of the last operation is also visible on the SystemVerilog Netlister window.

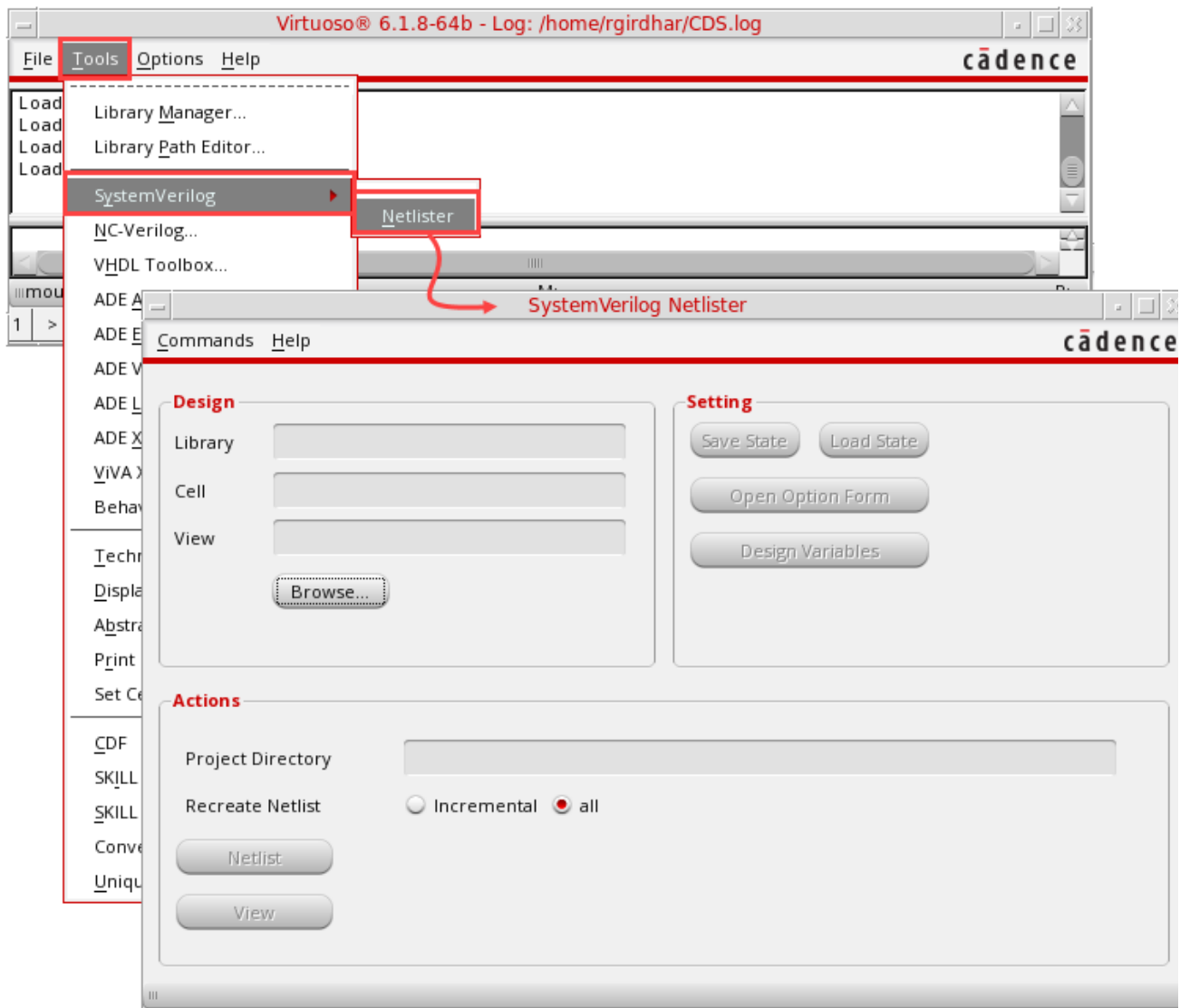
## Launching the SystemVerilog Netlister Interface

SystemVerilog Netlister provides a simple and efficient interface to let you configure settings and options for netlist generation.

To launch the SystemVerilog Netlister application and specify the design:

1. Launch Virtuoso.
2. In the CIW, choose *Tools – SystemVerilog – Netlister*.

The *SystemVerilog Netlister* window appears. Click *Browse* to specify the design.





## Virtuoso SystemVerilog Netlister User Guide

### Introducing Virtuoso SystemVerilog Netlister

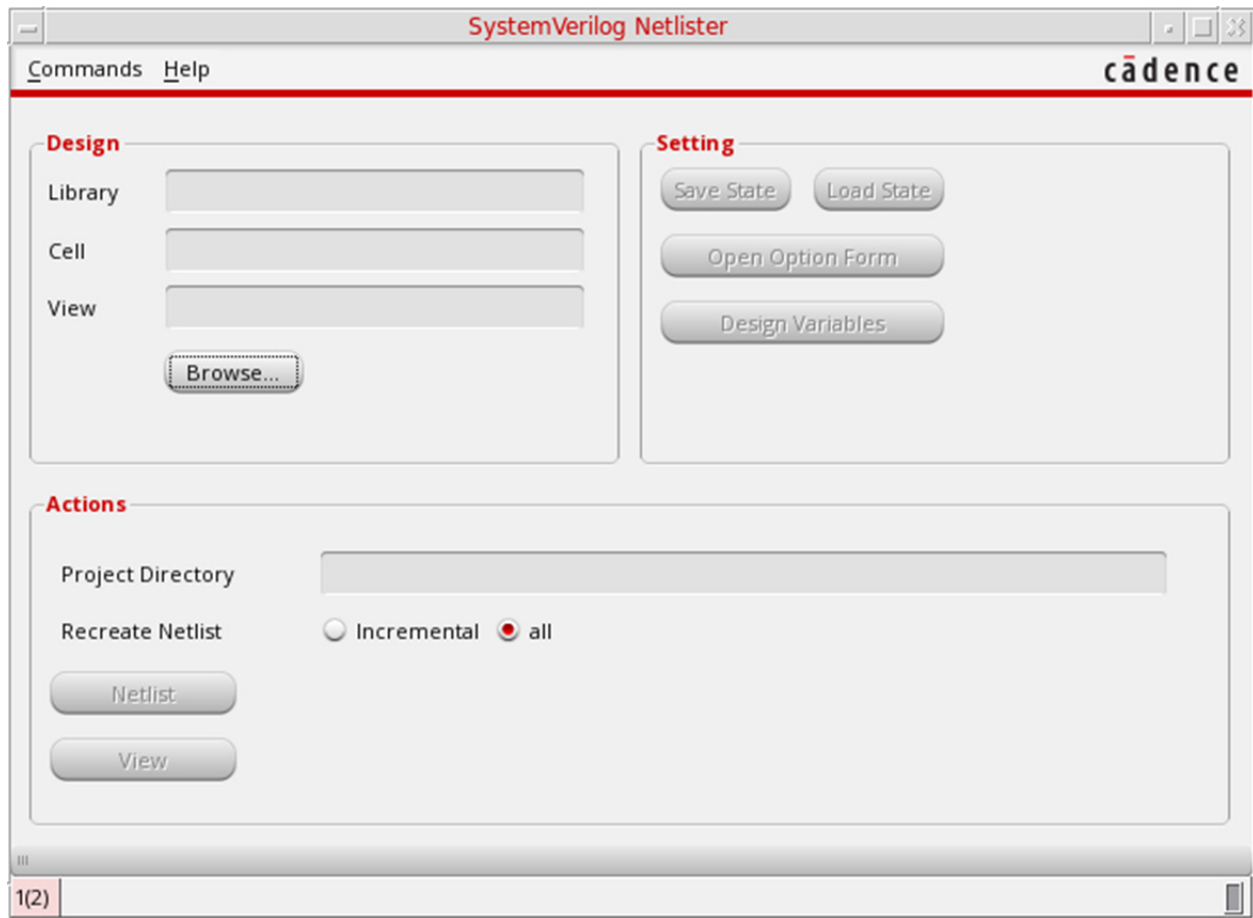
---

To select the design and launch the SystemVerilog Netlister application:

1. Launch Virtuoso.
2. In the CIW, choose *Tools – Library Manager*.  
The *Library Manager* window appears.
3. In the *Library Manager* window, select a library from the *Library* list.  
The cells present in the specified library appear in the *Cell* list.
4. Select a cell from the *Cell* list.  
The views present in the specified cell appear in the *View* list.
5. Select a `dnl_state*` view from the *View* list.  
The *SystemVerilog Netlister* window appears and shows the design.

## Understanding the Graphical User Interface

The following figure illustrates the main window of SystemVerilog Netlister:



The following table describes the main sections of the SystemVerilog Netlister window:

Section	Description
Menus	Displays the Commands and Help menus.
Design	Specifies the library, cell, and view of the top-level design. <b>Note:</b> You can select only configuration views. For details, see <a href="#">Specifying a Design</a> .

## Virtuoso SystemVerilog Netlister User Guide

### Introducing Virtuoso SystemVerilog Netlister

---

Section	Description
Setting	<p>Specifies the options and design variables required for netlist generation.</p> <p>For details, see <a href="#">Setting Up Netlist Generation Options</a> and <a href="#">Setting Up Design Variables</a>.</p>
Actions	<p>Specifies the netlisting actions. You can generate or regenerate a netlist to view the netlisting results.</p> <p>For details, see <a href="#">Netlisting a Design</a>.</p>
Status bar	<p>Displays the current status of the selected item.</p>

# **Virtuoso SystemVerilog Netlister User Guide**

## Introducing Virtuoso SystemVerilog Netlister

---

---

## Using SystemVerilog in Batch Mode

---

### Important

Before you launch SystemVerilog Netlister ensure that you meet all the prerequisites. See [Prerequisites for Using SystemVerilog Netlister](#).

SystemVerilog Netlister can be launched in batch mode by running the `runsv` command. This command supports the following options:

```
-lib <libName>
-cell <cellName>
-view <config_viewname>
-rundir <projectdir>
-netlist
-cdsenv <path of .cdsenvfile>
-state <saved SystemVerilog Netlister state (stateLib/stateCell/stateView)>
```

For example, if you have a design `topLib/topCell/config_sv` and you specify the project directory as `runsv_run`, use the following command to generate the netlist in batch mode:

```
runsv -lib topLib -cell topCell -view config_sv -netlist -rundir runsv_run
```

### Important

You can use `-cdsenv` optionally. However, if you want to enable data type propagation or create an additional argument file, set the following environment variables in the `.cdsenv` file:

- `enableDataPropagate`
- `defaultNettype`
- `mergedNetlist`
- `createXrunArgs`

The `-cdsenv` option lets you read the file. Alternatively, you can specify these environment variables in the `.cdsinit` file.

## Creating Config Views of SystemVerilog Designs

When using SystemVerilog Netlister, ensure that your specified design has a config view that contains only digital content. This digital content can be a SystemVerilog or Verilog text view, a schematic view, or a symbol view. If your design does not have a config view, you can use the `cdsCreateConfig` utility. Cadence provides this utility with SystemVerilog Netlister to help you create a config view.

You can use the following options with the `cdsCreateConfig` utility:

```
cdsCreateConfig -lib topLibName -cell topCellName -view topViewName [-config  
newConfigName] [-liblist 'lib1 lib2'] [-viewlist 'view1 view2'] [-stoplist  
'stopview1 stopview2']
```

For example, if you have a design `topLib/topCell/schematic`, use the following command in the terminal window to create a config view:

```
cdsCreateConfig -lib topLib -cell topCell -view schematic
```

To know more about `cdsCreateConfig`, type `cdsCreateConfig -help` in the terminal window for more details.

## Migrating SystemVerilog Integration Designs to SystemVerilog Designs

You can convert commands used in the Virtuoso Verilog Environment for SystemVerilog Integration Environment (SI) to a format supported by SystemVerilog Netlister by using the `si2runsv` utility.

You run SystemVerilog Integration netlister by using the following command:

```
si [run directory] [-batch [-command commandName]]
```

For example:

```
si dir1 -batch -command netlist
```

Alternatively, SystemVerilog Netlister uses the following command:

```
runsv -lib lib -cell cell -view configView -netlist -state stateName
```

To convert an SystemVerilog Integration environment command to a SystemVerilog Netlister command:

1. Run the following command to generate a script file:

```
si2runsv
```

This command generates a `runsv_xxxfile` script file.

## Virtuoso SystemVerilog Netlister User Guide

### Using SystemVerilog in Batch Mode

---

2. Run the following command to run SystemVerilog Netlister using this script file:

```
si2runsv dir1 -batch -command netlist
```

To know more about using `si2runsv`, use the following command:

```
si2runsv -help
```

# **Virtuoso SystemVerilog Netlister User Guide**

## Using SystemVerilog in Batch Mode

---



---

## Managing Netlist Generation

---

This chapter describes how to use SystemVerilog Netlister to netlist a design and manage states.

It contains the following topics:

- Netlisting a Design
  - Specifying a Design
  - Setting Up Netlist Generation Options
  - Setting Up Design Variables
  - Generating a Netlist
  - Viewing a Netlist
- Managing States
- Customizing Netlist Generation Using .simrc
- Adding Port Properties to an Instance

## Netlisting a Design

You can generate a netlist, which contains connectivity information of a design, after you have specified the design. Configure the netlist generation options before you generate the netlist. When you generate the netlist, SystemVerilog Netlister creates a netlist file of your design based on the settings that you specify and lets you view the netlist file.

This section contains the following topics:

- [Specifying a Design](#)
- [Setting Up Netlist Generation Options](#)
  - [Importing a SystemVerilog Package File](#)
  - [Specifying Additional Arguments](#)
- [Setting Up Design Variables](#)
- [Generating a Netlist](#)
- [Viewing a Netlist](#)

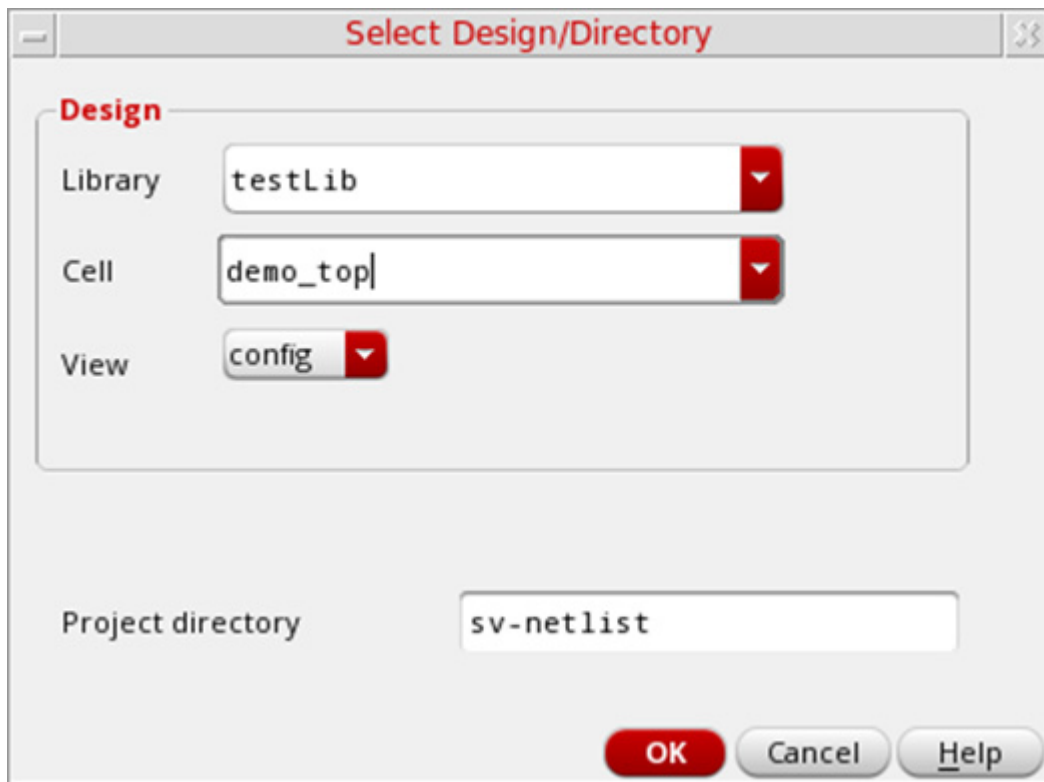
### Specifying a Design

You can start netlist generation by specifying a design.

To specify a design:

1. Open the SystemVerilog Netlister window. See [Launching the SystemVerilog Netlister Interface](#).
2. Click *Browse* in the *Design* group box.

The *Select Design/Directory* window appears.



3. Select a library from the *Library* list.

The cells in the specified library appear in the *Cell* list.

4. Select a cell from the *Cell* list.

The views in the specified cell appear in the *View* list.

5. Select a config view from the *View* list.

6. Specify a new project directory name in the *Project directory* field, if required.

The default project directory name is `sv-netlist`.

**Note:** The project directory contains various subdirectories. When you select a library, cell, and view in the *Select Design/Directory* window, a new subdirectory is created in the project directory. The subdirectory derives its name from the library, cell, and view names that you select in the design. For example, the typical directory structure of a design is as follows:

```
projectDir/lib_cell_view/netlist
```

7. Click *OK* to select the design.

The CIW displays an appropriate message to indicate that the design specification is successful.

## Setting Up Netlist Generation Options

You can set various options, based on which the SystemVerilog Netlister generates a netlist.

To configure netlist generation options:

1. Open the SystemVerilog Netlister window.

See [Launching the SystemVerilog Netlister Interface](#).

2. In the *Settings* group box, click *Open Option Form* to set additional options.

The SystemVerilog Netlister Options form appears.

## Virtuoso SystemVerilog Netlist User Guide

### Managing Netlist Generation

3. The *Netlist* tab of the SystemVerilog Netlist Options form appears as follows:

The screenshot shows the 'SystemVerilog Netlist Options' dialog box with the 'Netlist' tab selected. The 'Miscellaneous' tab is also visible. The 'Netlist' tab contains the following options:

- Port Declaration Format: ☒ Non-ANSI ☐ ANSI
- Default Nettype: ☒ interconnect ☐ wire
- Enable Datatype Propagation: ☐
- Enable Array Type Propagation Only: ☐
- Enable auto package importing: ☐
- Merge text source to single netlist: ☐
- netTypes to ignore on schematic nets:
- supply0:
- supply1:
- Pre-Module Include File:  ...
- In-Module Include File:  ...

At the bottom of the dialog are buttons for **OK**, **Cancel**, **Defaults**, **Apply**, and **Help**.

Specify values as follows:

Field	Description
<i>Port Declaration Format</i>	Select one of the following port declaration formats: <ul style="list-style-type: none"><li>■ Non-ANSI (default)</li><li>■ ANSI</li></ul> <p>When <code>hnlPrintNonAnsiSV</code> is set to <code>nil</code> in the <code>.simrc</code> file and datatype propagation is enabled, port declaration can be done only in ANSI format.</p>

## Virtuoso SystemVerilog Netlister User Guide

### Managing Netlist Generation

---

<i>Default Nettype</i>	<p>Select one of the following net types to netlist the nets on the schematic:</p> <ul style="list-style-type: none"><li>■ <i>interconnect</i>(default): If no net type is propagated, the net and the bus are declared explicitly as <i>interconnect</i>.</li><li>■ <i>wire</i>: A net without another explicitly set type is not declared. Here, only the bus is declared. Single-bit wires are not saved in the declaration because Xcelium resolves signals that are not explicitly declared to be of the type <i>wire</i>.</li></ul>
<i>Enable Datatype Propagation</i>	<p>Select the check box to enable data type propagation. It is not selected by default.</p> <p>Selecting this check box allows the propagation of both packed or unpacked, and datatype or portKind properties.</p>
<i>Enable Array Type Propagation</i>	<p>Select the check box to enable array type propagation. It is not selected by default.</p> <p>Selecting this check box disables the <i>Enable Datatype Propagation</i> check box and allows recognition of packed and unpacked arrays. As a result, the SystemVerilog Netlister prints a bus, connected to a port of the type <code>real</code> or a net type, as unpacked, and propagates the unpacked property up to top level. However, the datatype of the port is not propagated.</p>
<i>Enable auto package importing</i>	<p>Select this check box to enable the auto-package handling feature. It allows SystemVerilog Netlister to automatically search and find the SystemVerilog package files on which the <code>systemverilog</code> modules in the design depend. It is not selected by default. See <a href="#">Importing a SystemVerilog Package File</a>.</p> <p>Alternatively, you can manually pre-compile a package into a library that you define.</p>

## Virtuoso SystemVerilog Netlister User Guide

### Managing Netlist Generation

---

<i>Merge text source to single netlist</i>	<p>Select this check box to merge the text files to create a single and self-contained <code>netlist.sv</code> file. This file includes the netlist from the schematic and the original source files from the text view.</p> <p>If you select the <i>Merge text source to single netlist</i> check box, the contents of the <code>cds_alias.sv</code> or the <code>cds_globals.sv</code> file are printed in the <code>netlist.sv</code> file.</p> <p>However, deselecting the <i>Merge text source to single netlist</i> check box results in the following:</p> <ul style="list-style-type: none"> <li>■ The absolute path of <code>cds_alias.sv</code> is printed in <code>textInputs</code> but removed from the <code>netlist.sv</code> file.</li> <li>■ The absolute path of <code>cds_global.sv</code> is printed in <code>xrunArgs</code> but removed from the <code>netlist.sv</code> file.</li> <li>■ When <code>simVerilogGenerateSingleNetlistFile</code> is set to <code>t</code> in the <code>.simrc</code> file, it generates a single merged netlist, including <code>cds_alias.sv</code> and <code>cds_globals.sv</code>, if these files exist. When set to <code>nil</code>, it generates a split netlist.</li> <li>■ When <u>Pre-Module Include File</u> or <u>In-Module include File</u> and <i>Merge text source to single netlist</i> are selected, the contents of <i>Pre-Module Include File</i> or <i>In-Module Include File</i> are saved in <code>netlist.sv</code>.</li> </ul>
<i>supply0</i>	Specify a value to set the global ground net. This value is saved in the <code>cds_globals.sv</code> file.
<i>supply1</i>	Specify a value to set the global power net. This value is saved in the <code>cds_globals.sv</code> file.
<i>Pre-Module Include File</i>	<p>Specify 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>

## Virtuoso SystemVerilog Netlister User Guide

### Managing Netlist Generation

---

---

#### *In-Module include File*

Specify 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.

If `hn1VerilogDumpIncludeFilesInNetlist` is set to t, the content of the include file is copied to the netlist, instead of an ``include` statement.

Flag: `vlogifInModuleIncludeFile`

---

Observe the following points:

- ❑ The *supply0* and *supply1* options support only global signals in the design. Cross-checking with the schematic is not supported.
- ❑ SystemVerilog Netlister supports customizing netlist generation using the `.simrc` file. However, SystemVerilog Netlister might not always support all the variables in `.simrc` that are supported by the SI Netlister.
- ❑ Datatype propagation and default net type selection result in the following scenarios:
  - When *Default Nettype* is set to *Interconnect* and *Enable Datatype Propagation* is deselected, the interconnects are printed in the netlist in the Non-ANSI format but the datatype is not propagated.
  - When *Default Nettype* is set to *Interconnect* and *Enable Datatype Propagation* is selected, the interconnects, ports, and datatypes are printed in the netlist in the ANSI format.
  - When *Default Nettype* is set to *Wire* and *Enable Datatype Propagation* is selected, interconnects are not printed in the design but the ports are printed with the datatype in the netlist in the ANSI format.



## Virtuoso SystemVerilog Netlister User Guide

### Managing Netlist Generation

---

4. Select the *Miscellaneous* tab of the SystemVerilog Netlister Options form. The form appears as follows:

The screenshot shows the 'SystemVerilog Netlister Options' dialog box with the 'Miscellaneous' tab selected. The dialog has two tabs: 'Netlister' and 'Miscellaneous'. The 'Miscellaneous' tab contains the following options:

- hdl.var file**: A text input field with a browse button (...).
- Pre-compiled libraries (-reflib)**: A text input field with a browse button (...).
- Create arguments file**: A checkbox.
- Create binding files for xrun only**: A checkbox.
- Print Global Time Scale**: A checkbox.
- Additional Arguments**: A large text area for entering additional command-line arguments.

At the bottom of the dialog are five buttons: **OK** (highlighted in red), **Cancel**, **Defaults**, **Apply**, and **Help**.

Specify values as follows:

---

Field	Description
<i>hdl.var file</i>	<p>Specify the name of an <code>hdl.var</code> file to include the simulation options.</p> <p>To know more about the <i>hdl.var</i> file, see the <a href="#">Xcelium XRUN User Guide</a>.</p>

---

## Virtuoso SystemVerilog Netlister User Guide

### Managing Netlist Generation

---

---

#### *Pre-compiled libraries (-reflib)*

Specify the name of a pre-compiled library. If you have a package definition, you can pre-compile the package into the same library or different libraries.

**Note:** To manage text views of a design, pre-compile all the packages into a library. You can use `-reflib` to include all the libraries that contain the pre-compiled packages required in the design.

---

#### *Create arguments file*

Select this check box to create the `xrunArgs` file and other standard binding files. This file includes `netlist.sv`, `config.sv`, `textInputs`, binding files, and other files, such as `cds_globals.sv` and `cds_alias.sv`. These files are generated during netlisting and are needed for simulations.

---

#### *Create binding files for xrun only*

Select this check box only if you use `xrun`. Selecting this check box automatically selects the *Create arguments file* check box and creates two sets of binding files in the netlist directory: `xrunArgs`, `xrunArgs_vy` (compatible with Xcelium), and the `hdl.var`, `cds_xrun.lib` and `lib.map` files.

Both `xrunArgs` and `xrunArgs_vy` include the following:

- `xrun argument vcfg_inst_precedence`
- `xrun argument compcnfg`
- `config.sv` file
- `textInputs` file

In addition:

- `xrunArgs` includes the `lib.map` file.
- `xrunArgs_vy` includes the `hdl.var` and `cdslib` or `cds_xrun.lib` file.

If you specify an `hdl.var` file in the *Miscellaneous* tab of the SystemVerilog Netlister Options form, this file is included in the `hdl.var` file that SystemVerilog Netlister generates.

**Note:** You do not need to select this check box if you use a third-party simulation tool.

---

## Virtuoso SystemVerilog Netlister User Guide

### Managing Netlist Generation

---

---

#### *Print Global Time Scale*

Select the check box to print the global time scale. By default, SystemVerilog Netlister does not print the time scale globally.

When *Print Global Time Scale* is enabled, you can set the following variables in the `.simrc` file to overwrite the time values or units defined within a design:

- `simVerilogSimTimeValue` and `simVerilogSimTimeUnit`: Sets the Global Sim Time and Unit
- `simVerilogSimPrecisionValue` and `simVerilogSimPrecisionUnit`: Sets the Global Sim Precision and Unit

---

#### *Additional Arguments*

Click the button to specify additional `xrun` arguments. See [Specifying Additional Arguments](#).

---

#### 5. Click *OK*.

Before you generate the netlist of a design, ensure that you configure the netlist generation options, as required. SystemVerilog Netlister generates the netlist based on how you configure these options. SystemVerilog Netlister stores the `netlist` file in the `<projectDir>/<lib>_<cell>_<view>/netlist/` directory.

For example: `projectDir/lib_cell_view/netlist/netlist.sv`

### Importing a SystemVerilog Package File

Before you import SystemVerilog package files into your design for netlisting, ensure that your design has a package file, for example, `global_package.sv`, and a systemVerilog view that has imported this package file.

To import a package:

#### 1. In the Library Manager window:

- a. Create a new `systemVerilogPackage` view by choosing *File – New – Cell View*.  
The New File form opens.
- b. Specify the library and cell name in their respective fields.
- c. In the *View* list, select `systemVerilogPackage`.

- d. Click *OK*.

The New File form closes, and the *View* list in the Library Manager shows the new view.

- e. Double-click the `systemVerilogPackage` view.

The view opens with an empty package module in Virtuoso Text Editor.

**2. In the Virtuoso Text Editor window:**

- a. Choose *File – Open*.

The Open File form opens.

- b. Select the `global_package.sv` package file and open it in a *new tab*.

- c. Copy the contents of the package file and paste them into the package module of the `systemVerilogPackage` view.

- d. Click the *Check and Save* button on the toolbar.

Correct any errors that are reported.

- e. Open the `systemVerilog` view in a *new tab*.

This is the view that has imported the package file.

- f. Click the *Check and Save* icon on the toolbar.

Ensure that no errors are reported.

**3. In the SystemVerilog Netlister window:**

- a. In the *Setting* group box, click *Open Option Form*.

This opens the SystemVerilog Netlister Options form and shows the Netlister tab.

- b. On the Netlister page, select *Enable auto package importing*.

- c. Click *OK* to close the SystemVerilog Netlister Options form.

- d. In the Action group box, click *Netlist*.

Ensure that no errors are reported.

### Specifying Additional Arguments

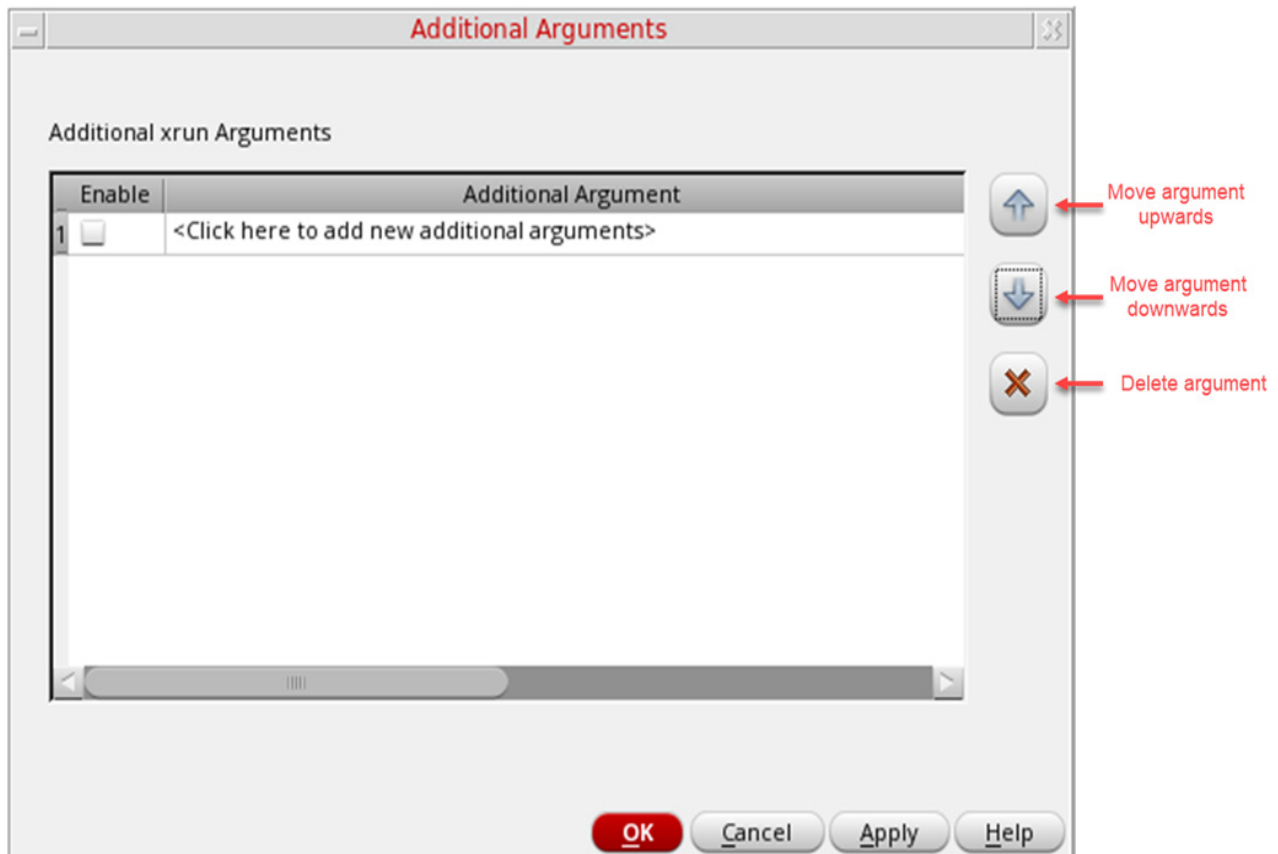
To specify additional `xrun` arguments in the netlisting settings in SystemVerilog Netlister:

1. In the SystemVerilog Netlister window, click *Browse* to specify a design.

The fields in the window are populated with the design details.

2. Click *Open Options Form* to access the netlisting settings.
3. In the Open Options Form window, click the *Miscellaneous* tab.
4. In the Miscellaneous page, click *Additional Arguments*.

The Additional Arguments form opens.



## Virtuoso SystemVerilog Netlister User Guide

### Managing Netlist Generation

---

The following table describes the columns available in the Additional Arguments form:

Column	Description
<i>Enable</i>	Enables the selected <code>xrun</code> argument.
<i>Additional Argument</i>	Specifies the name of the <code>xrun</code> argument. Click the field to add a new argument name or edit an existing argument name.

5. Specify valid `xrun` arguments in the Additional Arguments form.

These arguments are appended to the `xrunArgs` file that is generated. If *Create binding files for xrun only* is enabled, SystemVerilog Netlister appends these additional arguments to both the `xrunArgs` and `xrunArgs_vy` files.

6. Click OK.

The Additional Arguments form closes.

## Setting Up Design Variables

You can add design variables to your netlisting settings in SystemVerilog Netlister after choosing your design.

To set up a design variable:

## Virtuoso SystemVerilog Netlister User Guide

### Managing Netlist Generation

---

1. Click *Design Variables* in the SystemVerilog Netlister window.  
The Editing Design Variables form opens.

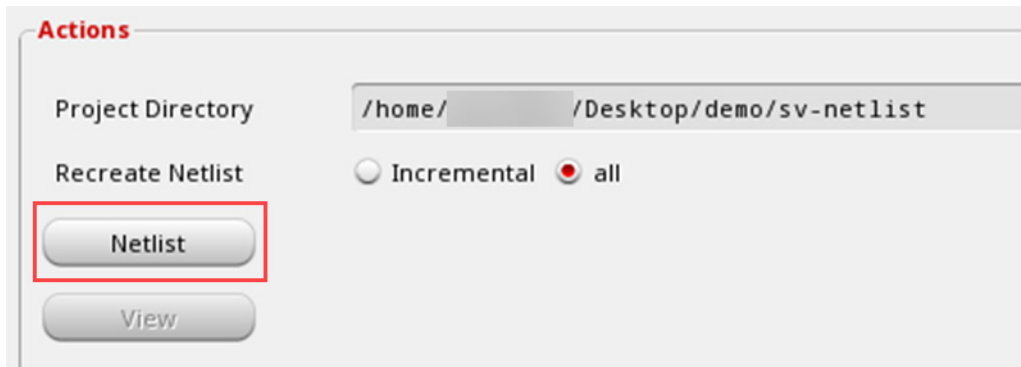
The screenshot shows the 'Editing Design Variables' dialog box. It features a title bar with the text 'Editing Design Variables'. The dialog is split into two panes. The left pane, labeled 'Selected Variable', has two text boxes for 'Name' and 'Value (Expr)', and three buttons: 'Add', 'Delete', and 'Change'. The right pane, labeled 'Design Variables', contains a table with headers 'Name' and 'Value'. At the bottom, there are four buttons: 'OK' (in a red box), 'Cancel', 'Apply', and 'Help'.

2. Specify a name for the design variable in the *Name* field.
3. Specify a value for the design variable in the *Value (Expr)* field.
4. Do the following:
  - ☐ To add a variable, click *Add*.  
The design variable appears in the *Design Variables* list on the right and is saved in the `cds_globals.sv` file.
  - ☐ To delete a design variable, select the variable name from the *Design Variables* list box and click *Delete*.
  - ☐ To change the name or value of a design variable, select the variable name from the *Design Variables* list box and click *Change*.  
Edit the name or the value of the design variable, as required.
5. Click *OK*.

## Generating a Netlist

To generate the netlist of a selected design:

1. Having set up options and variables for the selected design, click *Netlist* in the Actions group box of the SystemVerilog Netlister window.



SystemVerilog Netlister does the following:

- ❑ Generates the netlist and stores it in the `<projDir>/<lib>_<cell>_<view>/netlist` directory.
- ❑ Enables the *View* button to allow viewing the netlist file.

**Note:** By default, SystemVerilog Netlister creates the `netlist.sv` file in the `./sv-netlist/<lib>_<cell>_<view>/netlist` directory.

To regenerate the netlist:

- ➔ In the *Actions* group box, select one of the following:
  - ❑ *Incremental*: Netlists only the changes that you make to the design or the settings. In case of large designs, this mode of netlist generation highly improves the performance.
  - ❑ *All*: Recreates the complete netlist. In case of large designs, this mode of netlist generation might decrease the performance.

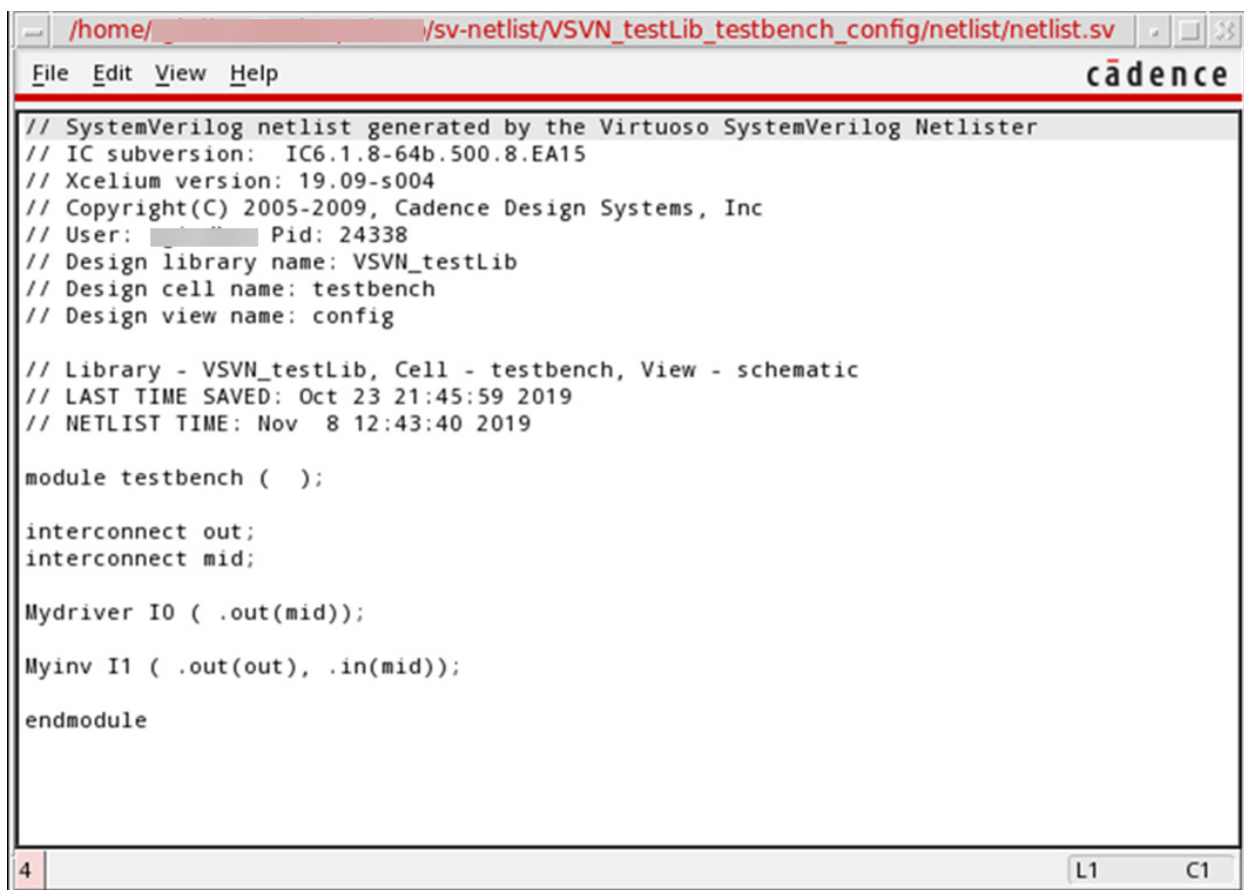


## Viewing a Netlist

A netlist contains the connectivity information of a design. To view the netlist of your design:

- ➡ In the *Actions* group box of the SystemVerilog Netlister window, click *View*.  
The `netlist.sv` file opens.

The following figure illustrates how SystemVerilog Netlister displays the netlist file:



The screenshot shows a text editor window titled `/home/.../sv-netlist/VSVN_testLib_testbench_config/netlist/netlist.sv` with the Cadence logo in the top right corner. The window contains the following SystemVerilog netlist code:

```
// SystemVerilog netlist generated by the Virtuoso SystemVerilog Netlister
// IC subversion: IC6.1.8-64b.500.8.EA15
// Xcelium version: 19.09-s004
// Copyright(C) 2005-2009, Cadence Design Systems, Inc
// User: [redacted] Pid: 24338
// Design library name: VSVN_testLib
// Design cell name: testbench
// Design view name: config

// Library - VSVN_testLib, Cell - testbench, View - schematic
// LAST TIME SAVED: Oct 23 21:45:59 2019
// NETLIST TIME: Nov 8 12:43:40 2019

module testbench ( );

interconnect out;
interconnect mid;

Mydriver IO ( .out(mid));

Myinv I1 ( .out(out), .in(mid));

endmodule
```

The status bar at the bottom shows line 4, column 1 (L1 C1).

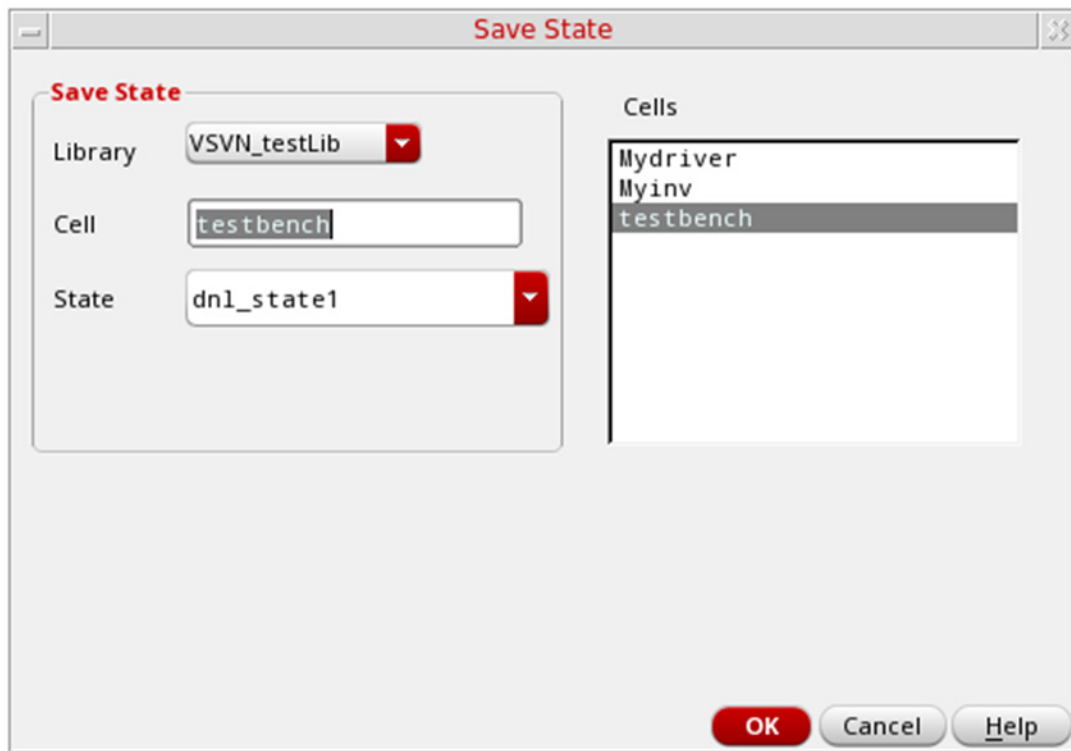
## Managing States

You can save the current state of your settings or load saved states and settings in SystemVerilog Netlister. It is useful when you want to avoid repeated setups of netlisting settings.

## Saving States

To save a state with the specified settings:

1. In the *Settings* group box of the SystemVerilog Netlister window, click *Save State*. The Save State form appears.

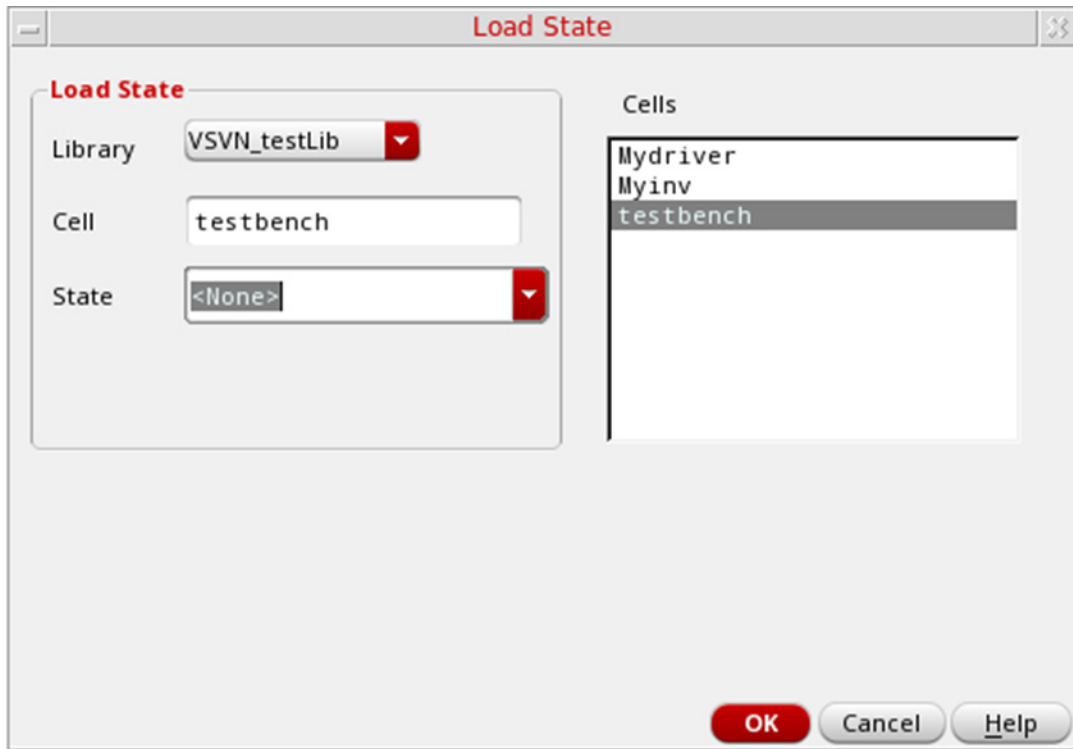


2. In the *State* field, specify a new name for the current state.
3. Click *OK*.

## Loading States

To load a saved state when you launch SystemVerilog Netlister:

1. In the *Settings* group box of the SystemVerilog Netlister window, click *Load State*. The Load State form appears.



2. In the *State* list, select the name of the state that you want to load.
3. Click OK.

## Customizing Netlist Generation Using .simrc

You can customize the netlist generation in SystemVerilog Netlister by setting the following variables in the .simrc file:

Variable	Description
hnlGetSimulator	Returns SystemVerilog Netlister for SystemVerilog Netlister.
hnlUserShortCVList	<p>Allows specifying a shorting list that contains devices that need to be shorted.</p> <p>Example: <code>hnlUserShortCVList=list(list(&lt;libName&gt; &lt;cellName&gt;))</code></p>
simVerilogEnableEscapeNameMapping	<p>Includes escaped names in the netlist. It also allows you to escape names that are reserved keywords in SystemVerilog.</p> <p>Default value is <code>t</code>.</p> <p>Example: If you have a module "assign" in your design, and you set <code>simVerilogEnableEscapeNameMapping</code> to <code>t</code>, it is mapped to <code>"\assign "</code> in the netlist.</p>
simSVPortPropertyList	<p>When enabled, allows specifying the <code>dataType</code> and <code>portKind</code> properties on a symbol cell. For example:</p> <pre>simSVPortPropertyList = '( ("analogLib" "res" "symbol" "PLUS real var" "MINUS wrealdriver nil") )</pre> <p>Here,</p> <ul style="list-style-type: none"><li>■ "analogLib" indicates the library name</li><li>■ "res" indicates the cell name</li><li>■ "symbol" indicates the view name</li><li>■ "PLUS real var" indicates that <code>dataType</code> is set to <code>real</code>, and <code>portKind</code> is set to <code>var</code> on port <code>PLUS</code>;</li><li>■ "MINUS wrealdriver nil" indicates that only the <code>dataType</code> property is set to <code>wrealdriver</code> on port <code>MINUS</code>, leaving the <code>portKind</code> property blank.</li></ul>

## Virtuoso SystemVerilog Netlister User Guide

### Managing Netlist Generation

Variable	Description
<code>vlogExpandIteratedInst</code>	<p>When set to <code>nil</code>, allows instances in the array in the following format:</p> <pre>"I_iter[0:2]"</pre> <p>When set to <code>t</code>, allows splitting iterated instances in the following format:</p> <pre>I_iter_1 ... I_iter_2 ... I_iter_3 ...</pre>
<code>hnlVerilogDumpIncludeFilesInNetlist</code>	<p>When set to <code>t</code>, copies the content of an included HDL file directly to the netlist, instead of inserting an <code>`include</code> statement.</p> <p>When this variable is set to <code>t</code>, the content of the text cellviews in the design hierarchy are copied to the netlist. Any file specified in the <i><u>Pre-Module Include File</u></i> or <i><u>In-Module include File</u></i> option of the SystemVerilog Netlister is also copied to the netlist.</p> <p><b>Notes:</b></p> <ul style="list-style-type: none"><li>■ This variable works only in single netlist file mode.</li><li>■ This variable does not support recursive inclusion of text files. Consider that <code>fileA.sv</code> includes <code>fileB.sv</code>. If you copy the contents of <code>fileA.sv</code> in the netlist using this variable, the contents of <code>fileB.sv</code> will not be copied in the netlist.</li></ul> <p>Default: <code>nil</code></p>
<code>simVerilogGenerateSingleNetlistFile</code>	<p>A flag to generate a single Verilog netlist containing multiple modules instead of one netlist per module. By default, this flag is set to <code>nil</code>. If set to <code>t</code>, the netlister generates a single Verilog netlist file in the current simulation run directory with the name <code>netlist</code>.</p>

## Virtuoso SystemVerilog Netlister User Guide

### Managing Netlist Generation

---

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

---

To know more about using the `.simrc` file, see [Virtuoso ADE Explorer User Guide](#).

## Adding Port Properties to an Instance

The SystemVerilog Netlister 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 Netlister 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.

## Virtuoso SystemVerilog Netlister User Guide

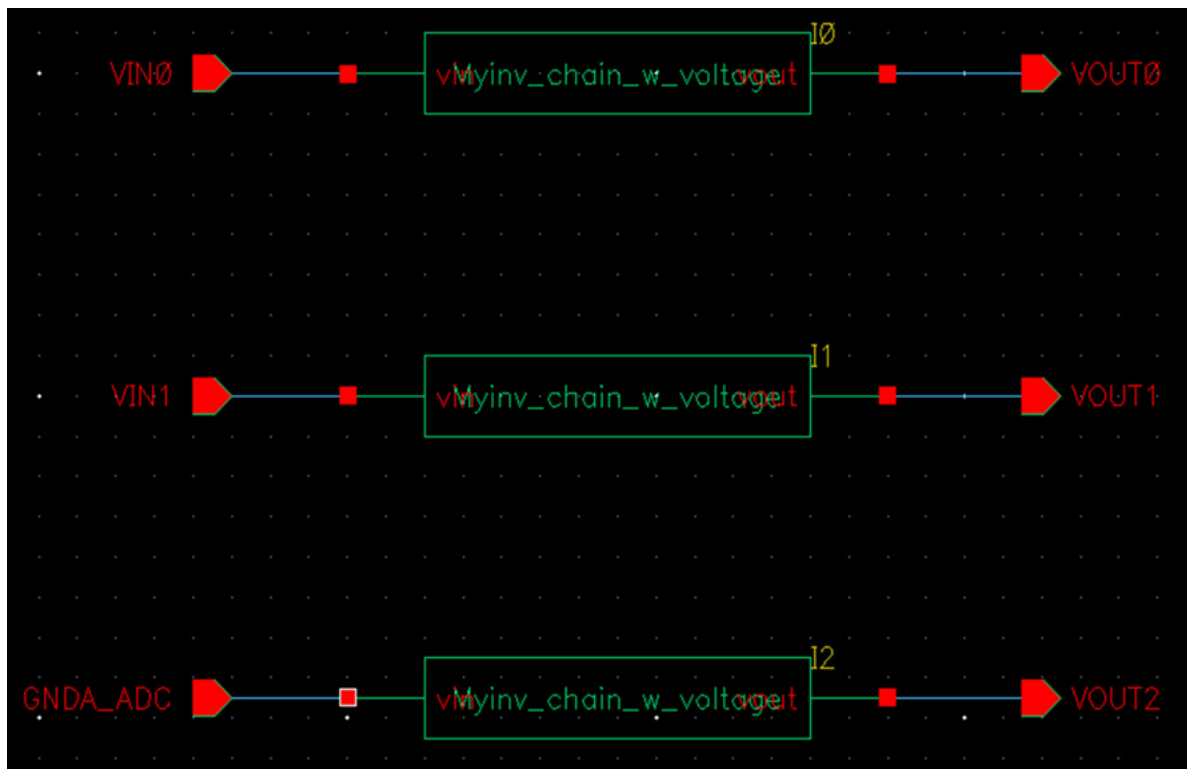
### Managing Netlist Generation

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 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:  value

Direction:

Signal Type:

Net Name:

Buttons: Add, Delete, Modify

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

Buttons: **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 );
```



## Virtuoso SystemVerilog Netlist User Guide

### Managing Netlist Generation

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.

**Edit Object Properties**

Apply To:

Show: ☒ user

Name:

Direction:

Signal Type:

Net Name:

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

## Virtuoso SystemVerilog Netlister User Guide

### Managing Netlist Generation

```
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.

User Property	Master Value	Local Value	Display
dataType	voltage		off
isRefPort	false		off
portKind			off
ignoreDataType			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 voltage GNDA_ADC,  
input voltage VIN0,  
input voltage VIN1 );
```

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

# **Virtuoso SystemVerilog Netlist User Guide**

## **Managing Netlist Generation**

---

# **Virtuoso SystemVerilog Netlist User Guide**

## **Managing Netlist Generation**

---

---

# Environment Variables

---

This appendix chapter describes the environment variables that control the characteristics of the SystemVerilog Netlister environment. These include:

- isPortInANSIFormat
- defaultNettype
- enableDataPropagate
- mergedNetlist
- vlogSupply0Sigs
- vlogSupply1Sigs
- hdlVarFile
- refLib
- createXrunArgs
- createXrunBinding
- enableTimeScale

## **isPortInANSIFormat**

Controls if the port declaration is in the *Non-ANSI* or *ANSI* format.

In `.cdsenv`:

```
digitalSim.netlisterOpts isPortInANSIFormat 'string "Non-ANSI"
```

In `.cdsinit`:

```
envSetVal("digitalSim.netlisterOpts" "isPortInANSIFormat" 'string  
"ANSI")
```

Valid Values:

Non-ANSI	Specifies that the port declaration is in the <i>Non-ANSI</i> format.
ANSI	Specifies that the port declaration is in the <i>ANSI</i> format.

Default Value: Non-ANSI

GUI Field      *Port Declaration Format*

## **defaultNettype**

Sets the default net type to netlist nets on the schematic.

In `.cdsenv`:

```
digitalSim.netlisterOpts defaultNettype 'string "interconnect"
```

In `.cdsinit`:

```
envSetVal("digitalSim.netlisterOpts" "defaultNettype" 'string  
"wire")
```

Valid Values:

"interconnect"	Sets the default net type to <i>interconnect</i> .
"wire"	Sets the default net type to <i>wire</i> .

Default Value: "interconnect"

GUI Field      *Default Nettype*

## **enableDataPropagate**

Controls the datatype propagation in SystemVerilog Netlister.

In `.cdsenv`:

```
digitalSim.netlisterOpts enableDataPropagate 'boolean t
```

In `.cdsinit`:

```
envSetVal("digitalSim.netlisterOpts" "enableDataPropagate"  
          'boolean nil)
```

Valid Values:

t	Enables the datatype propagation.
nil	Disables the datatype propagation.

Default Value: `nil`

GUI Field     *Enable Datatype Propagation*

## **mergedNetlist**

Merges the text files to create a single and self-contained netlist. This netlist includes the netlist from the schematic and the original source files from the text view.

In `.cdsenv`:

```
digitalSim.netlisterOpts mergedNetlist 'boolean t
```

In `.cdsinit`:

```
envSetVal("digitalSim.netlisterOpts" "mergedNetlist" 'boolean nil)
```

Valid Values:

t	The text files are merged to a single netlist successfully.
nil	The text files are not merged to a single netlist.

Default Value: `nil`

GUI Field     *Merge text source to single netlist*

## **vlogSupply0Sigs**

Sets the global ground net.

In `.cdsenv`:

```
digitalSim.netlisterOpts vlogSupply0Sigs 'string ""
```

In `.cdsinit`:

```
envSetVal("digitalSim.netlisterOpts" "vlogSupply0Sigs" 'string "")
```

Valid Values:

A valid ground net value.

Default Value: ""

GUI Field      *Supply0*

## **vlogSupply1Sigs**

Sets the global power net.

In `.cdsenv`:

```
digitalSim.netlisterOpts vlogSupply1Sigs 'string ""
```

In `.cdsinit`:

```
envSetVal("digitalSim.netlisterOpts" "vlogSupply1Sigs" 'string "")
```

Valid Values:

A valid power net value.

Default Value: ""

GUI Field      *Supply1*



## hdlVarFile

Allows you to select the `hdl.var` file from a specific location. To know more, see the [\*Xcelium XRUN User Guide\*](#).

In `.cdsenv`:

```
digitalSim.compilerOpts hdlVarFile 'string ""
```

In `.cdsinit`:

```
envSetVal("digitalSim.compilerOpts" "hdlVarFile" 'string "")
```

Valid Values:

Path to the directory where the `hdl.var` file is located.

GUI Field     *Hdl.var file*

## refLib

Allows the selection of a pre-compiled library. If you have a package definition, you can pre-compile the package into a single library or multiple libraries.

In `.cdsenv`:

```
digitalSim.compilerOpts refLib 'string ""
```

In `.cdsinit`:

```
envSetVal("digitalSim.compilerOpts" "refLib" 'string "")
```

Valid Values:

Path to the directory where the `refLib` file is located.

GUI Field     *Pre-compiled libraries (-reflib)*

## createXrunArgs

Controls the default setting for the *Create arguments file* check box in the SystemVerilog Netlister Options form.

In `.cdsenv`:

```
digitalSim.compilerOpts createXrunArgs 'boolean t
```

In `.cdsinit` or the CIW:

```
envSetVal("digitalSim.compilerOpts" "createXrunArgs" 'boolean nil )
```

Valid Values:

t	The <i>Create arguments file</i> check box in the SV Netlister Options form is selected by default.
nil	The <i>Create arguments file</i> check box in the SV Netlister Options form is deselected by default.

Default Value: `nil`

GUI Field      *Create arguments file*

## createXrunBinding

Creates the `hdl.var` and `cds_xrun.lib` files in the netlist directory when you use `xrun`.

In `.cdsenv`:

```
digitalSim.compilerOpts createXrunBinding 'boolean t
```

In `.cdsinit` or the CIW:

```
envSetVal("digitalSim.compilerOpts" "createXrunBinding" 'boolean  
nil )
```

Valid Values:

t	The files are created successfully.
nil	The files are not created.

Default Value: `nil`

GUI Field      *Create binding files for xrun only*

## **enableTimeScale**

Enables printing the global time scale.

In `.cdsenv`:

```
digitalSim.elabOpts enableTimeScale 'boolean t
```

In `.cdsinit` or the CIW:

```
envSetVal("digitalSim.elabOpts" "enableTimeScale" 'boolean nil )
```

Valid Values:

<code>t</code>	The global time scale is printed successfully.
----------------	--

<code>nil</code>	The global time scale is not printed.
------------------	---------------------------------------

Default Value: `nil`

GUI Field     *Print Global Time Scale*

## **nettypesToIgnore**

Controls the net types to be ignored.

In `.cdsenv`:

```
digitalSim.netlisterOpts nettypesToIgnore 'string ""
```

In `.cdsinit` or the CIW:

```
envSetVal("digitalSim.netlisterOpts" "nettypesToIgnore" 'string ""  
)
```

Valid Values:

<code>t</code>	The nettype is ignored successfully.
----------------	--------------------------------------

<code>nil</code>	The nettype is not ignored.
------------------	-----------------------------

Default Value: `nil`

GUI Field     *Print Global Time Scale*

## **Virtuoso SystemVerilog Netlist User Guide**

### **Environment Variables**

---

---

# SKILL Functions

---

This section provides syntax, descriptions, and examples for the Cadence SKILL functions associated with the Virtuoso SystemVerilog Netlister (SystemVerilog Netlister) flow.

Only the functions documented in this chapter are supported for public use. Any other functions, and undocumented aspects of the functions described below, are private and subject to change or removal at any time.

## asiDigitalSimAutoloadProc

```
asiDigitalSimAutoloadProc(  
    )  
=> t / nil
```

### Description

Automatically loads the context for class and tool initialization when you launch SystemVerilog Netlister.

### Argument

None

### Value Returned

t	Returns t, if the function call is successful.
nil	Returns nil, if the function is unsuccessful.

### Example

The following example shows using the function:

## Virtuoso SystemVerilog Netlist User Guide

### SKILL Functions

---

```
asiDigitalSimAutoloadProc ( )  
=> t
```