cadence®

Virtuoso Import Tools User Guide

Product Version ICADVM20.1
October 2020

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

Contents

Preface

This document provides information about the tools that you can use to import analog, digital, and mixed-signal netlist files into the Virtuoso Design Environment.

Readers should be familiar with using the command-line interface in the Virtuoso design environment.

This preface contains the following topics:

Licensing Requirements

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

Related Documentation

What's New and KPNS

- *Virtuoso Import Tools What's New*
- *Virtuoso Import Tools Known Problems and Solutions*

Installation, Environment, and Infrastructure

- *Cadence Installation Guide*
- *Virtuoso Software Licensing and Configuration User Guide*
- *Virtuoso Design Environment User Guide*
- *Cadence Application Infrastructure User Guide*
- *Virtuoso Design Environment SKILL Reference*

Technology Information

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

Virtuoso Tools

- *Virtuoso Text Editor User Guide*
- *Verilog In for Virtuoso Design Environment User Guide and Reference*
- *VHDL In for Virtuoso Design Environment User Guide and Reference*
- *Connectivity to Schematic User Guide*
- *HDL Import and Netlist-to-Schematic Conversion SKILL Reference*

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.

To explore the full range of training courses provided by Cadence in your region, visit Cadence Training or write to training_enroll@cadence.com.

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

Help and Support Facilities

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

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

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

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

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

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

- Log on to Cadence Online Support

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

Feedback about Documentation

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

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

You can also submit feedback by using the following methods:

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

Typographic and Syntax Conventions

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

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

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

1

Import Tools in the Virtuoso Design Environment

Import tools in the Virtuoso Design Environment let you import analog, digital, and mixed-signal netlists into the Virtuoso Design Environment. Importing a file includes converting the information it contains to a syntax that Virtuoso can understand.

Import tools can be classified on the basis of the language used to describe the design information:

- Analog import tools – spiceIn, Virtuoso Text Editor, cdsTextTo5x

  Used for designs written in languages such as SPICE, HSPICE, PSPICE, SPECTRE, and CDL

- Digital import tools – Verilog In, VHDL In, Virtuoso Text Editor, cdsTextTo5x

  Used for designs written in languages such as Verilog, SystemVerilog, and VHDL. Here, Verilog In and VHDL In are language dependent.

- Mixed-Signal import tools – Text Editor, cdsTextTo5x

  Used for designs written in languages such as Verilog AMS and VHDL AMS.

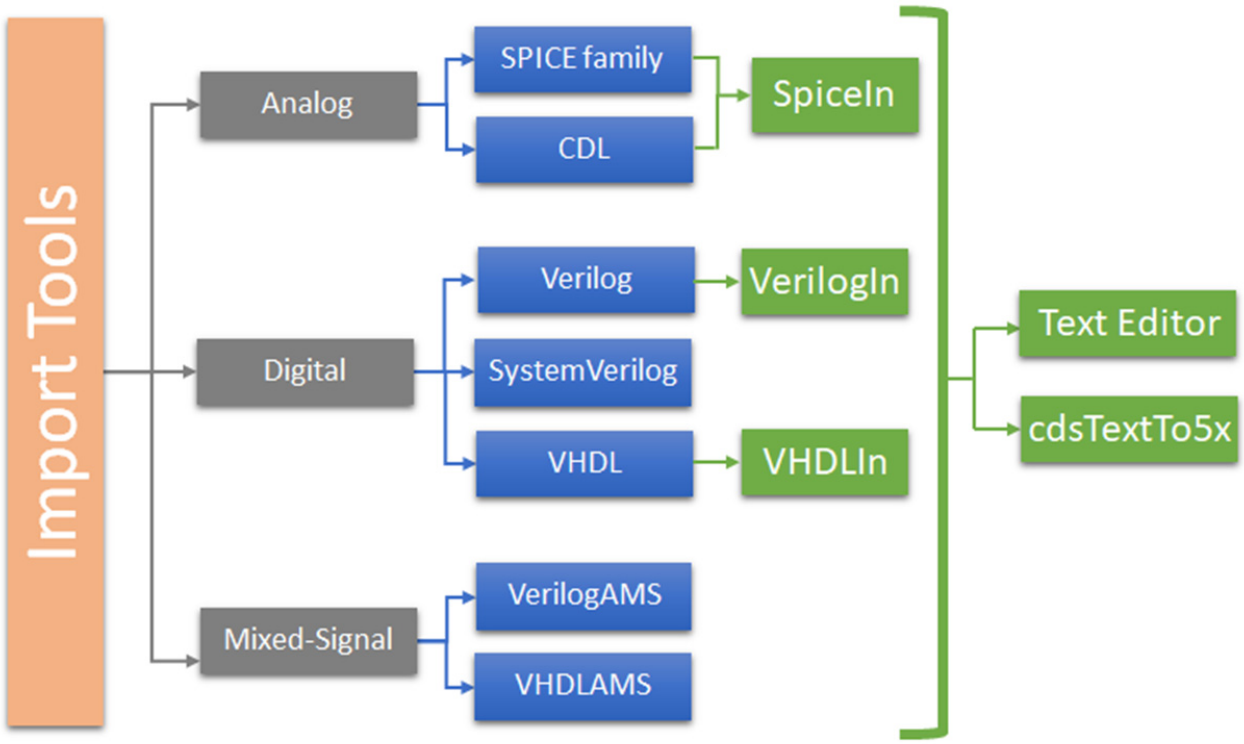
All import tools generate the following elements:

- The 5x library structure – The library, cell, and view structure is generated to import the netlist file of the design, for example `verilog.v`, into the Virtuoso Design Environment environment without the need to launch Virtuoso.
- A symbol view of the specified cell – The symbol representation of the cell interface can then be instantiated as a place master in a schematic.

- A shadow database of netlist files – These tools save OpenAccess connectivity information in the `netlist.oa` file. Shadow-based netlisters need the `netlist.oa` file to ensure that the same connectivity information exists in the place master and the switch master.
- A schematic view – These tools generate a schematic view of the design.

  Note: cdsTextTo5x does not support creation of schematic views.
- A blackbox of a cell – These tools optionally allow importing only the interface information of a design cell while ignoring the instance information. This improves the temporal performance when you import large designs.
- A single cell – These tools allow importing a specific cell or set of cells from a netlist file. This functionality eliminates the need to import the complete netlist.

The following illustration shows the import tools hierarchy with the Virtuoso import tools shown in green:



The import tools and the languages that they support are as follows:

- Analog design files which are written using SPICE, PSPICE, SPECTRE, DSPF, or CDL are imported into the DFII environment by using the spiceIn, Text Editor, or cdsTextTo5x tools.
- Digital design files which are written using Verilog and VHDL, are imported using Verilog In and VHDL In, respectively, because these tools are language dependent. These files can also be imported using the Text Editor and cdsTextTo5x tools.
- Digital design files which are written using SystemVerilog and the mixed-signal design files written using Verilog AMS and VHDL AMS are imported using the Test Editor and cdsTextTo5x tools.

Most import tools generate schematics, symbols, and the OpenAccess database. The exception is cdsTextTo5x, which generates only symbols and the OpenAccess database.

For more information, see:

- *Verilog In for Virtuoso Design Environment User Guide and Reference*
- *VHDL In for Virtuoso Design Environment User Guide and Reference*
- *Connectivity to Schematic User Guide*
- *HDL Import and Netlist-to-Schematic Conversion SKILL Reference*
- *Virtuoso Text Editor User Guide*

2

Importing Design Data by Using cdsTextTo5x

The cdsTextTo5x tool is a small and powerful command-line binary that lets you import analog, digital, and mixed-signal text files of designs written in various languages into the Virtuoso Design Environment. It supports information written in SPICE, CDL, Verilog, SystemVerilog, Verilog AMS, VHDL, or VHDL AMS.

This chapter contains the following topics:

Benefits of cdsTextTo5x

Importing netlist files into the Virtuoso Design Environment involves launching Virtuoso, creating the schematics, and then creating the database. An ideal solution would be to create a very small binary which accepts a netlist file and some settings, generates the required symbols and the OpenAccess database, and imports the netlist file into the Virtuoso Design Environment.

The cdsTextTo5x binary can import any analog, digital, or mixed-signal netlist information file. It does not use the traditional DPL-based mechanism but instead uses an mAPI or Verification Procedural Interface (VPI)-based engine.

Note: Verification Procedural Interface (VPI) is an interface used primarily for the C programming language.

When you use a VPI-based engine, the shadow database is generated more quickly. In comparison, a DPL-based engine that is used by the Virtuoso Text Editor utilizes more system resources and is more time-consuming.

For analog languages, shadow database generation is fast because these languages do not depend on DPL-based engines. The dependency on DPL exists only for digital languages. Using a VPI engine removes the high dependency on DPL engines and removes the involved capacity limitations.

cdsTextTo5x produces the same results as the Text Editor. It imports a design and then creates the corresponding symbols and shadows 10 times faster than the Text Editor. Based on Cadence benchmarks, cdsTextTo5x imports 20M pin with 10M instances in an hour as compared to 10 hours with the Text Editor. Additionally, cdsTextTo5x also supports importing the cell as a blackbox when you want to import only the interface information.

Important

Performance results may vary depending on design size and hardware configuration.

The main features of cdsTextTo5x are:

- Supports the most commonly used netlist description languages
- Increases the import time performance of Verilog-based designs
- Imports a cell as a blackbox when you want to import only the interface information
- Provides full shadow support, which includes instance information
- Removes dependency on Virtuoso to generate symbols

- Starts by importing the instance master. If the master definition does not exist, it does not import the cell and exits with appropriate error messages.
- Allows faster parsing to import only the specified cell if multiple module definitions exist in the same text file
- Eliminates the use of SKILL-based information and increases performance and supportability with reduced resources

For more information, see:

- Comparison between cdsTextTo5x and Virtuoso Text Editor
- Performance of cdsTextTo5x

Comparison between cdsTextTo5x and Virtuoso Text Editor

The following table compares the functionality of the Text Editor and cdsTextTo5x:

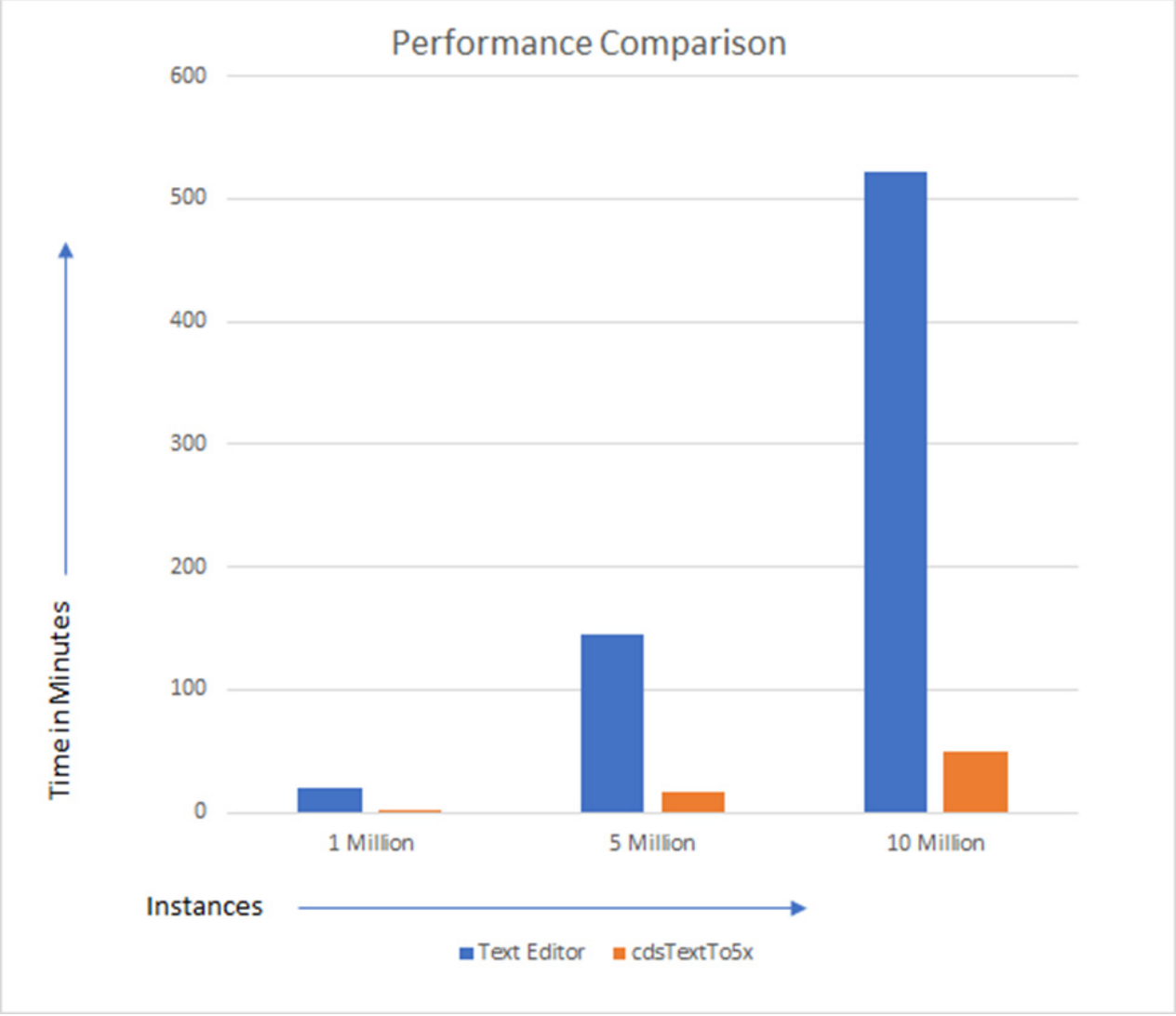
Feature	Function	Text Editor	cdsTextTo5x
Supports all analog languages	Imports files written in Spice, HSpice, PSpice, Spectre, DSPF, and CDL languages.	Yes	Yes
Supports all digital languages	Imports files written in Verilog, SystemVerilog, and VHDL languages.	Yes	Yes
Supports all mixed-signal languages	Imports files written in VerilogAMS and VHDLAMS languages.	Yes	Yes
Eliminates dependency on the deprecated DPL mechanism (Better performance)	DPL is an intermediate representation of a parsed netlist file which is a SKILL list of lists, but it is outdated. The recommended mechanism is using VPI to read data directly from the parsed file.	No	Yes
Eliminates dependency on the deprecated `-use5x` command line option (Better performance)	The `-use5x` option is not scalable and is outdated. It is only available from the parser to generate the library, cell, and view structure.	No	Yes
Integrates library definitions through cds.lib	Eliminates the need to specify reference libraries and their paths manually in the parameter file.	Yes	Yes

Feature	Function	Text Editor	cdsTextTo5x
Connectivity information in the shadow database in case of implicit connections	Mandates the shadow database to have the following connectivity information, if the instance line in the netlist file has implicit connectivity. ■ Implicit connectivity. For example, resistor R0 (a,b). ■ Explicit connectivity. For example, resistor R0(.PLUS(a), .MINUS(b)).	Yes	Yes
Blackbox support	Allows importing only the top level interface information of a cell that might contain millions of instances. The interface information is imported without processing the instances.	No	Yes
Creation of symbolic link to netlist file	Creates a symbolic link to file in the library/cell/view structure without importing the actual netlist file	No	Yes

Performance of cdsTextTo5x

Consider a scenario where a Verilog file contains a cell definition that comprises 1 million, 5 million, and 10 million instances whose master (symbols) exist in a reference library.

The following illustration shows the difference in performance of cdsTextTo5x as compared to Text Editor when evaluated on an exclusive system with 90GB RAM:



Important

Currently, cdsTextTo5x supports these performance improvements only for Verilog-based cells.

Sample results and performance in the above scenarios are as follows:

- Instances: 1 million
  - Time taken by Text Editor: 19min 10sec
  - Time taken by cdsTextTo5x: 2min 40sec
  - Comparative performance of cdsTextTo5x: **6x** faster
- Instances: 5 million
  - Time taken by Text Editor: 144min 5sec, approximately 2hr 30 min
  - Time taken by cdsTextTo5x: 17min 10sec
  - Comparative performance of cdsTextTo5x: **8x** faster
- Instances: 10 million, and 20 million vector pins at the top level
  - Time taken by Text Editor: 522min, approximately 8hr 42min
  - Time taken by cdsTextTo5x: 50min
  - Comparative performance of cdsTextTo5x: **10x** faster

Important

Performance results may vary depending on design size and hardware configuration.

The results above clearly indicate that cdsTextTo5x delivers faster and better performance when importing analog, digital, or mixed-signal netlist files.

Working with the cdsTextTo5x Command

This section includes the following topics:

Prerequisites

Before you run `cdsTextTo5x`, ensure that Virtuoso, `xrun`, and text file parsers including `xmvlog` and `xmvhdl` are set correctly in the terminal from which you want to run the command.

Running the cdsTextTo5x Command

Use the following `cdsTextTo5x` command syntax:

```
$ cdsTextTo5x ARGUMENTS FILE_TO_IMPORT
```

For example, to import a Verilog file called `test.v` in the library `myLib`, run the following command:

```
$ cdsTextTo5x -LIB myLib test.v
```

To import a Verilog file called `test.v` in the `myLib`/`myCell`/`myView` cellview and create its symbol view `mySymbol`, run the following command:

```
$ cdsTextTo5x -LANG verilog -LIB myLib -CELL myCell -VIEW myView -SYMBOL mySymbol
test.v
```

The following table describes the arguments that you can specify when using the `cdsTextTo5x` command:

Argument	Description
`-LIB <library_name>`	Library where the cellview is generated. **Note:** This is a mandatory argument. Additionally, you must specify the name of the file you want to import after the library name.
`-CELL <cell_name>`	

Argument	Description
	Name of the cell for which the 5x structure is to be generated. **Note:** This is an optional argument.
`-VIEW <view_name>`	Name of the view. The command uses the default names if the cell and view names are not specified. **Note:** This is an optional argument.
`-SYMBOL symbol_view_name`	Name of the symbol view to be generated. The command does not create a symbol view if the symbol name is not specified. **Note:** This is an optional argument.
`-CDSLIB cds.lib_to_use`	Name of the `cds.lib` file with the path to the specified library. If you do not specify this value, the default `cds.lib` file is used. If the file does not exist, it is created. **Note:** This is an optional argument.
`-LANG language_name`	Language of the text file being imported. The valid values are `spice`, `pspice`, `dspf`, `spectre`, `verilog`, `verilogams`, `systemverilog`, `vhdl`, and `vhdlams`. If you do not specify the language, the command determines the language from the filename extension of the file being imported. **Note:** This is an optional argument.
`-LOG log_file_name`	Name of the log file. **Note:** This is an optional argument.

Argument	Description
-COMPILEROPTIONS *compiler_options_file_name*	
	File containing option settings to be passed directly to the compiler (irun or xrun) when cdsTextTo5x is executed. **Note:** This is an optional argument.
-SHADOW	Generates the shadows for Verilog or SystemVerilog views. This argument does not require any parameters. If shadow generation fails when using cdsTextTo5x, you can view the errors and warning messages in the cdsTextTo5x.log file. For analog languages, such as spice, pspice, dspf, and spectre, the OA database is generated by default. You do not need to specify the -SHADOW argument explicitly. **Note:** This is an optional argument.
-BLACKBOX	Imports only the interface information for a cell and does not include the instance information. This argument can be used only when -CELL is specified. Example: cdsTextTo5x -LIB *work* -CELL *bot* -BLACKBOX *verilog.v* **Note:** This is an optional argument. This functionality is supported for Verilog only.
-SPECIFICUNIT	Imports only the specified cell if multiple module definitions exist in the same text file. All other cells are ignored. The performance is improved when multiple cells exist in a file. In such cases, parsing saves a lot of time. This argument can be used only when -CELL is specified. **Note:** This is an optional argument.
-HELP	Displays the help for the command. Use this argument on its own. **Note:** This is an optional argument.

Examples

Example 1:

```
cdsTextTo5x -LIB myLib test.v
```

Imports the `test.v` file into the Virtuoso Design Environment and generates its library, cell, and view structure.

Example 2:

```
cdsTextTo5x -LIB myLib -SYMBOL symbol test.v
```

Imports the `test.v` file into the Virtuoso Design Environment and generates the symbol corresponding to the cell defined in the file.

Example 3:

```
cdsTextTo5x -LIB myLib -SHADOW test.v
```

Imports the `test.v` file into the Virtuoso Design Environment and generates the OpenAccess database (`netlist.oa`) corresponding to the cell defined in the file.

Example 4:

```
cdsTextTo5x -LIB myLib -SHADOW -CELL top -BLACKBOX test.v
```

Imports only the `top` cell and ignores the instance information. It fetches the interface information in the OpenAccess database only.

Example 5:

```
cdsTextTo5x -LIB myLib -CELL top -SPECIFICUNIT test.v
```

Imports only the `top` cell if multiple cells are defined in `test.v`.

3

Importing Design Data by Using Virtuoso Text Editor

Virtuoso® Text Editor lets you work with the text cellviews stored in HDL files, such as Verilog, SystemVerilog, VHDL, PSpice, HSPICE, Spectre, and SPICE text files, in Virtuoso libraries. You use this application to perform the following tasks:

- Create and edit digital and analog text cellviews.
- Check the syntax of text cellviews.
- Create text cellview databases.
- Create different types of views, such as symbol and schematic views, from a text cellview.
- Check the pin order in text cellviews.

This chapter contains the following topics:

Customization of Text Cellview by Using SKILL Variables

You can use variables to customize the operation of text cellviews. For example, you can put these variables in a `.cdsinit` file or set their values in the CIW.
The variables apply when you do one of the following:

- Type in or edit a text cellview
- Create a text cellview from another cell using one of the *Design – Create Cellview* commands from the schematic or symbol editor
- Create another cellview from a text cellview
- Call the function `vmsUpdateCellViews`

The following SKILL variables can be used to customize the behavior of text cellviews.

`vmsRunningInUI`	`vmsCrossViewCheck`
`vmsUpdatePcdb`	`vmsDoNotCheckMasterFileWritable`
`vmsSensitivityIssuesReport`	`vmsNcvlogExecutable`
`vhdlCreateEntityFromArch`	`vmsPortProcessing`
`vhdlCrossViewCheck`	`vmsTemplateScript`
`vhdlKeepCaseAsNC`	`vmsTemplateScriptForVerilog`
`vhdlIdentCaseSensitive`	`vmsTemplateScriptForSystemVerilog`
`vhdlUpdateSymbol`	`vmsUpdateSymbolAfterEdit`
`vmsAnalysisType`	`vmsVerboseMsgLevel`
`vmsCreateMissingMasters`	

vmsRunningInUI

Controls whether messages are displayed in dialog boxes.

```
vmsRunningInUI_variable ::=
        vmsRunningInUI = t | nil
```

The parameters are the following:

`t`	Messages are displayed in dialog boxes rather than in the CIW.
`nil`	Messages are displayed in the CIW.

vmsUpdatePcdb

Controls the update of the `pc.db` file of the source cellview while doing a Cellview to Cellview operation. By default, during a Cellview to Cellview operation, the `pc.db` file of the source cellview is not updated.

```
vmsUpdatePcdb_variable ::=
    vmsUpdatePcdb_variable = t | nil
```

The parameters are the following:

`t`	The `pc.db` file is updated automatically.
`nil`	The `pc.db` file is not updated automatically.

vmsSensitivityIssuesReport

Controls whether the tool reports the issues that occur when the sensitivity properties are applied to a terminal during text extraction. The possible values are - "`Summary`" , "`Detailed`" , "`Ignored`".

```
vmsSensitivityIssuesReport_variable ::=
    vmsSensitivityIssuesReport = Summary | Detailed | Ignored
```

The parameters are the following:

`Summary`	A generalized information is reported at the end of extraction if sensitivity related issues are detected during text view extraction. This is the default value.
`Detailed`	All warnings displayed during text view extraction are printed.

`Ignore`	All sensitivity related issues are ignored completely. The summary is not printed.

vhdlCreateEntityFromArch

For VHDL views, creates an entity view from the architecture when symbol view is present. The architecture view may be extracted from Virtuoso Text Editor or external editors. The parameters are the following:

`t`	The entity view is automatically created from architecture. This is the default value.
`nil`	Disables creation of entity views.

vhdlCrossViewCheck

Controls whether symbol views are checked for consistency with other views in a cell when the `vmsUpdateCellViews` function is run.

The parameters are the following:

`t`	Checks whether the symbol views that are created are consistent with all other views in a cell.
`nil`	Disables the consistency check.

vhdlKeepCaseAsNC

By default, names of VHDL identifiers (such as entity, port and architecture names) are lower cased when the `vmsUpdateCellViews` function is run. Use this variable to preserve the case of entity and port names when the `vmsUpdateCellViews` function is run.

Note the following:

- Architecture names are always lowercased.
- Cadence recommends that you do not use the following environment variable to preserve the case of entity and port names:

  ```
  CDS_ALT_NMP=match
  ```

For example, consider the following VHDL entity:

```
entity myEntity is
port(
VIn : In bit
Vout : out bit
);
```

When the vmsUpdateCellViews function is run, by default, the symbol view contains lower cased port names vin and vout. The entity name is also converted to lowercase and saved as myentity.

Note: In this case, the original VHDL text view does not get modified. Instead, a new VHDL entity view named myentity, is created.

The parameters are the following:

t	Case of entity and port names are preserved. For escaped names, case is preserved for all identifiers. **Note:** If t, you must also add the following entry in the hdl.var file: `define ncvhdlopts -keepcase4use5x`
nil	Names of VHDL identifiers are lowercased.

vhdlIdentCaseSensitive

Specifies whether names of objects, such as, pins, signals, aliases, and instance names are case sensitive.

The parameters are the following:

t	Object names are case sensitive.
nil	Object names are not case sensitive.

vhdlUpdateSymbol

Controls whether symbol views are automatically created for cells that don't have a symbol view when the vmsUpdateCellViews function is run.

The parameters are the following:

t	The symbol view is automatically created.
nil	Disables creation of symbol views.

query	Displays a pop-up asking for confirmation whether the symbol view should be created.

vmsAnalysisType

Specifies the kind of syntax checks to be applied to text views.

```
vmsAnalysisType_variable ::=
        vmsAnalysisType = "Analog" | "Digital" | "Mixed"
```

The parameters are the following:

Analog	The syntax in text views is checked for compliance with the Verilog-A language specification.
Digital	The syntax in text views is checked for compliance with the Verilog (digital-only) language specification. This is the default value for verilog text views.
Mixed	The syntax in text views is checked for compliance with the Verilog-AMS language specification. This is the default value for verilog-ams text views.

vmsCreateMissingMasters

Specifies whether the environment is to create skeleton descriptions for undefined modules.

```
vmsCreateMissingMasters_variable ::=
        vmsCreateMissingMasters = t | nil
```

The parameters are the following:

t	The environment creates skeleton Verilog-AMS descriptions and symbols for undefined modules by using explicit port names (when the instances are connected explicitly) or by using connecting module port names (when the instances are connected implicitly). If these approaches fail, the environment uses dummy names for ports. The direction assigned to each port is based on the direction of the connecting net, if a direction is set.
nil	The environment does not create skeleton descriptions or symbols for undefined modules. This is the default value.

vmsCrossViewCheck

Controls whether symbol views are checked for consistency with other views in a cell when the vmsUpdateCellViews function is run.

The parameters are the following:

`t`	Checks whether the symbol views that are created are consistent with all other views in a cell.
`nil`	Disables the consistency check.

Default value: `t`

vmsDoNotCheckMasterFileWritable

Specifies whether the program should check the master file for write privileges before performing the `vmsUpdateCellViews` function.

```
vmsDoNotCheckMasterFileWritable_variable ::=
        vmsDoNotCheckMasterFileWritable = t | nil
```

The parameters are the following:

`t`	In order to update cellviews using the `vmsUpdateCellViews` function when the source file is read-only, set this variable to `t`.
`nil`	If the master file is read-only, the `vmsUpdateCellViews` function will not perform the update. This is the default value.

vmsNcvlogExecutable

Specifies which ncvlog executable is to be used to parse the Verilog-AMS text file.

```
vmsNcvlogExecutable_variable ::=
        vmsNcvlogExecutable = "path_and_executable"
```

The parameter is the following:

path_and_executable

The executable used to parse the text file.

vmsPortProcessing

Determines how port concatenations are handled when the environment generates a Verilog-AMS text view from another cellview.

```
vmsPortProcessing_variable ::=
        vmsPortProcessing = "Analog" | "Digital" | "Mixed"
```

The parameters are the following:

`Analog`	Port concatenations remain as concatenations in the generated cellviews.
`Digital`	Port concatenations remain as concatenations in the generated cellviews. This is the default value for verilog text views when `schHdlUseVamsForVerilog` is set to `t`.
`Mixed`	Port concatenations in generated cellviews are translated to the format expected by the AMS netlister. This is the default value for verilog-ams text views. This is the default value for verilog text views when `schHdlUseVamsForVerilog` is set to `nil`.

Example

You have a symbol with two terminals named `a<2:3>,b,c<1>` and `c<2:3>,b`. If `vmsPortProcessing` is set to `Analog` or `Digital` and the terminals are of the inout type, the AMS Designer environment creates the following skeletal text module from the symbol.

```
module <name> ( {a[2:3], b, c[1]}, {c[2:3], b} );
    inout  [1:3] c;
    inout  b;
    inout  [2:3] a;
endmodule
```

If `vmsPortProcessing` is set to `Mixed`, the AMS Designer environment creates the following skeletal module, which is in the format expected by the AMS netlister.

```
module <name> ( a, b, c )
    inout  [1:3] c;
    inout  b;
    inout  [2:3] a;
endmodule
```

vmsTemplateScript

Specifies the name of a script used to customize the header information for new Verilog-AMS cellviews.

```
vmsTemplateScript_variable ::=
        vmsTemplateScript = "template_script" | nil
```

The parameters are the following:

template_script

	The specified script is used to generate headers.
`nil`	A default header is used. It has the form `//Verilog-AMS HDL for` *`libname, cellname viewname`*

Example

Assume that `vmsTemplateScript` is set to `"template_script"` and `template_script` contains

```
#!/bin/csh -f
echo "// Verilog-AMS HDL for " \"$1\", \"$2\" \"$3\"
echo ""
echo "// Module    : $2"
echo "// Description   :"
echo "// First Created :" `date`
echo "//"
echo "//"
echo "// MODULE DESCRIPTION :"
echo "//"
echo "// EVENTS DESCRIPTION :"
echo ""
exit 0
```

Now assume that a new cell called `test`, with the view `vams`, is created in the library `vams_test_lib`. A new Verilog-AMS cell is generated with the following information:

```
// Verilog-AMS HDL for "vams_test_lib", "test" "vams"

// Module        : test
// Description   :
// First Created : Wed Apr 5 15:01:26 IST 2000
//
//
// MODULE DESCRIPTION :
//
// EVENTS DESCRIPTION :

module test ( );

endmodule
```

vmsTemplateScriptForVerilog

Specifies the name of a script used to customize the header information for new Verilog cellviews.

For more information, see the example in vmsTemplateScript.

```
vmsTemplateScriptForVerilog_variable ::=
        vmsTemplateScriptForVerilog = "template_script" | nil
```

The parameters are the following:

template_script
: The specified script is used to generate headers.

nil
: A default header is used. It has the form

  //Verilog HDL for *libname*, *cellname viewname*

vmsTemplateScriptForSystemVerilog

Specifies the name of a script used to customize the header information for new SystemVerilog cellviews.

For more information, see the example in vmsTemplateScript.

```
vmsTemplateScriptForSystemVerilog_variable ::=
        vmsTemplateScriptForSystemVerilog = "template_script" | nil
```

The parameters are the following:

template_script
: The specified script is used to generate headers.

nil
: A default header is used. It has the form

  //systemVerilog HDL for *libname*, *cellname viewname*

vmsUpdateSymbolAfterEdit

Controls whether symbol views are automatically created for cells that don't have a symbol view when the vmsUpdateCellViews function is run.

The parameters are the following:

t
: The symbol view is automatically created.

nil
: Disables creation of symbol views.

query
: Displays a pop-up asking for confirmation whether the symbol view should be created.

Default value: query

vmsVerboseMsgLevel

Specifies the highest message level to be printed. Higher numbers result in more messages being printed; lower numbers result in fewer messages being printed.

```
vmsVerboseMsgLevel_variable ::=
        vmsVerboseMsgLevel = message_level
```

The parameter is the following:

message_level

An integer equal to or greater than zero, which is the highest message level to be printed.

Level 0 (zero) messages are printed as is. Messages of level 1 or higher are prefixed with `VAMS Diagnostics:`

Messages are categorized as fatal (`F`), warning (`W`), or error (`E`) and each is displayed with a mnemonic. For example,

```
\o VAMS *W, MNEERR: Inherited Expressions for multiple member terms for port "zz"
\o        cannot be formatted. Declaring it without any net expression
\o        attribute.
```