cadence®

Virtuoso Import Tools User Guide

Product Version IC23.1
September 2023

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

Contents

1

Import Tools in the Virtuoso Studio Design Environment

Import tools in the Virtuoso® Design Environment let you import analog, digital, and mixed-signal netlists into the Virtuoso Studio 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 analog designs written in languages, such as SPICE, HSPICE, PSPICE, SPECTRE, and CDL.

- Digital import tools – VerilogIn, VHDLIn, Virtuoso Text Editor, cdsTextTo5x

  Used for digital designs written in languages, such as Verilog and VHDL. Here, VerilogIn and VHDLIn are language dependent. These files can also be imported using the Virtuoso Text Editor and cdsTextTo5x tools.

- Mixed-signal import tools – Virtuoso Text Editor, cdsTextTo5x

  Used for digital designs written using SystemVerilog and the mixed-signal design files written using Verilog AMS and VHDL AMS.

In addition to being language-specific, these tools also have their individual benefits and uses.

cdsTextTo5x

The cdstextTo5x binary lets you import analog, digital, and mixed-signal text files of designs written in various languages into the Virtuoso Studio Design Environment. It supports information written in SPICE, CDL, Verilog, SystemVerilog, Verilog AMS, VHDL, or VHDL AMS.

For more information, see Design Data Import by Using cdsTextTo5x.

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.

For more information, see:

Design Data Import by Using Virtuoso Text Editor

Virtuoso Text Editor User Guide

Verilog In

Verilog In lets you convert structural Verilog netlists into one of the following forms:

- Virtuoso schematics
- Netlists in the Virtuoso format
- Verilog text views in a Virtuoso library

For more information, see *Verilog In User Guide*.

VHDL In

VHDL In lets you convert a VHDL structural or behavioral description into one of the following forms in Cadence OpenAccess database (OA) storage format:

- Schematic view
- Netlist view
- VHDL text view

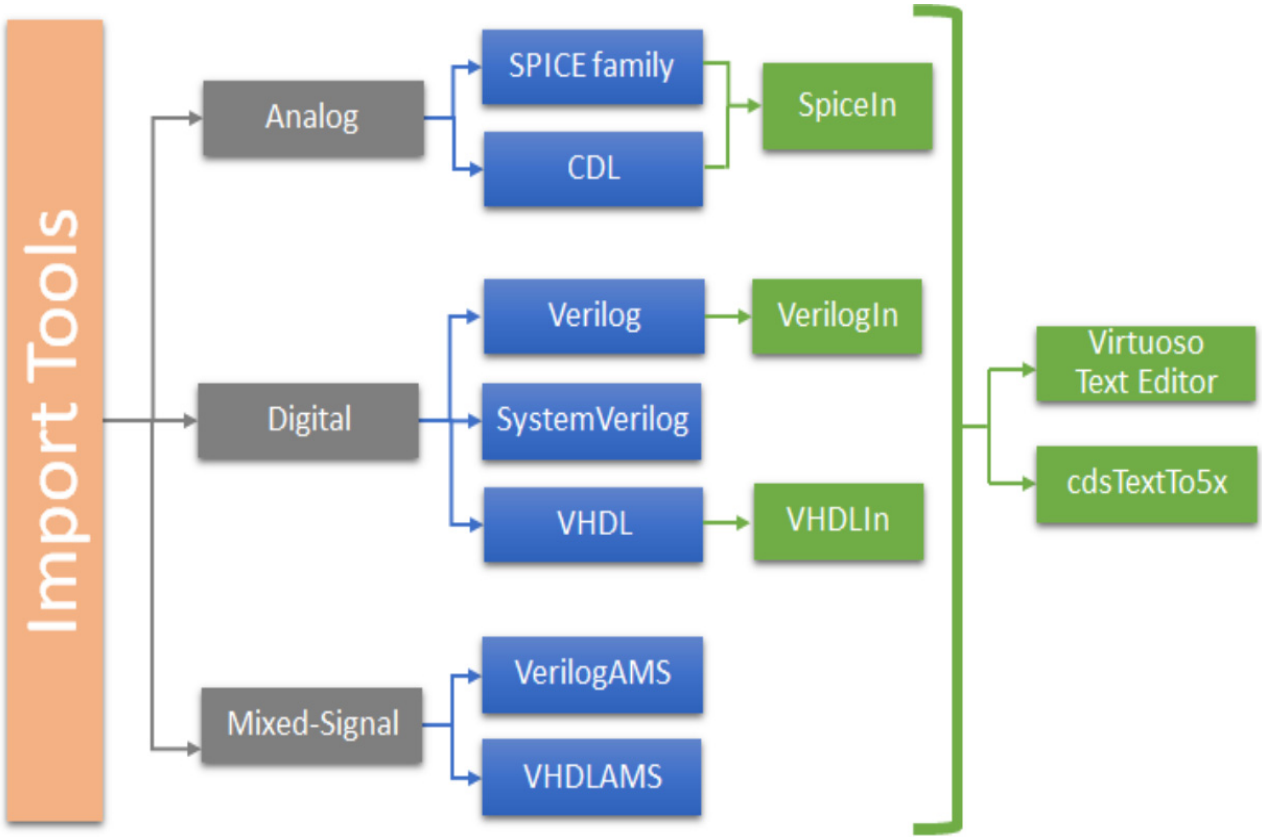
For more information, see *VHDL In for Virtuoso Studio Design Environment User Guide and Reference*

Connectivity to Schematic

Connectivity-to-Schematic (conn2sch) lets you generate schematic views from netlist views. A netlist view contains only connectivity information for a design. The Connectivity-to-Schematic tool generates a fully placed and routed schematic view using the connectivity information. You can create both, digital and analog schematic views using this tool. The Connectivity-to-Schematic tool provides different placement and routing engines to generate digital and analog schematic views. A schematic view can be opened, viewed, and edited using Virtuoso® Schematic Editor.

For more information, see *Connectivity to Schematic User Guide*

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



Related Topics

Design Data Import by Using cdsTextTo5x

Elements Generated by Import Tools

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 Studio 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.

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

Related Topics

Virtuoso Text Editor

Verilog In

VHDL In

Connectivity to Schematic

HDL Import and Netlist-to-Schematic Conversion SKILL Reference

Licensing Requirements

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

2

Design Data Import 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 Studio Design Environment. It supports information written in SPICE, Verilog, SystemVerilog, Verilog AMS, VHDL, or VHDL AMS.

Related Topics

Benefits of cdsTextTo5x

Comparison between cdsTextTo5x and Virtuoso Text Editor

Performance of cdsTextTo5x

Importing a Verilog File Using the cdsTextTo5x Command

cdsTextTo5x Arguments

Benefits of cdsTextTo5x

Importing netlist files into the Virtuoso Studio 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 Studio 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
- Supports Component Description Format (CDF) for analog and digital languages

Related Topics

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

Related Topics

Benefits of cdsTextTo5x

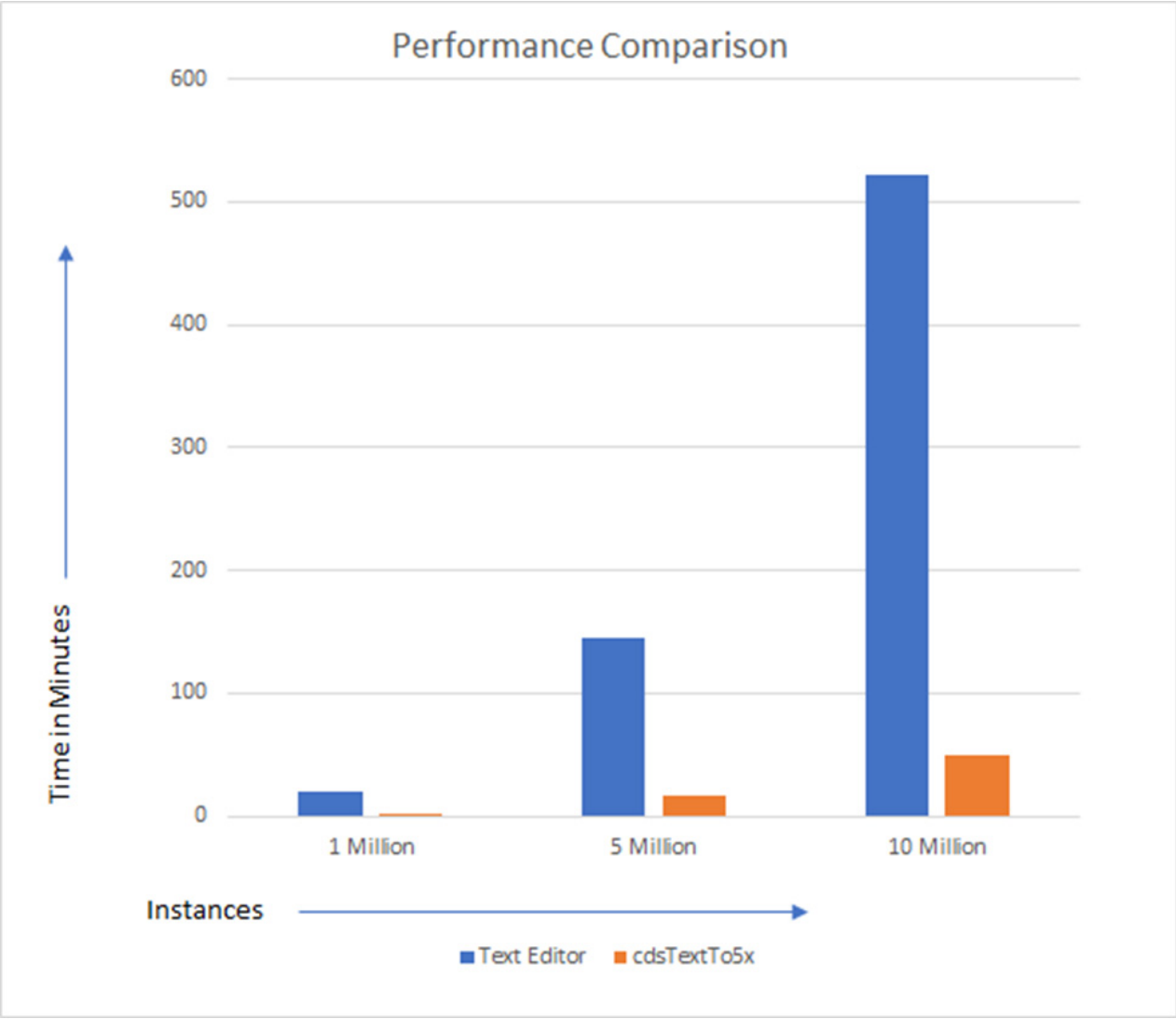
Design Data Import by Using cdsTextTo5x

Virtuoso Text Editor User Guide

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.

Related Topics

Design Data Import by Using cdsTextTo5x

Benefits of cdsTextTo5x

Comparison between cdsTextTo5x and Virtuoso Text Editor

Importing a Verilog File Using the cdsTextTo5x Command

Importing a Verilog File Using the cdsTextTo5x Command

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.

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
```

Related Topics

cdsTextTo5x Arguments

Design Data Import by Using cdsTextTo5x

Benefits of cdsTextTo5x

Comparison between cdsTextTo5x and Virtuoso Text Editor

Performance of cdsTextTo5x

cdsTextTo5x Arguments

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

Argument	Description
`-LIB <library_name>`	Name of the library where the cellview is generated. This is a mandatory argument. You must also specify the name of the file you want to import after the library name.
`-CELL <cell_name>`	Name of the cell for which the 5x structure is to be generated. This is an optional argument. The command uses the default name if the cell name is not specified.
`-VIEW <view_name>`	Name of the view. This is an optional argument. The command uses the default names if the cell and view names are not specified.
`-SYMBOL symbol_view_name`	Name of the symbol view to be generated. This is an optional argument. The command does not create a symbol view if the symbol name is not specified.
`-CDSLIB cds.lib_to_use`	Name of the `cds.lib` file with the path to the specified library. This is an optional argument. If you do not specify this value, the default `cds.lib` file is used. If the file does not exist, it is created.

Argument	Description
`-LANG` *`language_name`*	
	Language of the text file being imported. The valid values are `spice`, `pspice`, `dspf`, `spectre`, `verilog`, `verilogams`, `systemverilog`, `vhdl`, and `vhdlams`. This is an optional argument. If you do not specify the language, the command determines the language from the filename extension of the file being imported. If the language is specified, the command honors the language irrespective of the specified file extension. If the *`FILE_TO_IMPORT`* in the `cdsTextTo5x` command does not contain a file extension, specifying the `-LANG` argument becomes mandatory.
`-LOG` *`log_file_name`*	
	Name of the log file. The file also contains the instance binding report. This is an optional argument.
`-COMPILEROPTIONS` *`compiler_options_file_name`*	
	File containing option settings to be passed directly to the compiler (`irun` or `xrun`) when cdsTextTo5x is executed. 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. This is an optional argument. 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.
`-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`* This is an optional argument. This functionality is supported for Verilog only.

Argument	Description
-HDLPKGMODE	Enables the HDL Package mode. This argument can be used only when -COMPILEROPTIONS <*compiler_options_file_name*> is specified. Here, <*compiler_options_file_name*> contains the compiler (XRUN) arguments that have been exported from the HDL Package Setup form. Example: cdsTextTo5x -LIB *testlib* -HDLPKGMODE -COMPILEROPTIONS *options.f test.sv* This is an optional argument.
-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 is an optional argument. This argument can be used only when -CELL is specified.
-PARAM /[-P] *param_file_name*	Specifies the name of the parameter file. This is an optional argument.
-TEMPLATEPARAMFILE *template_param_file_name*	Creates a template parameter file with the specified name. This is an optional argument.
-HELP	Displays the help for the command. This is an optional argument.

Examples

Example 1

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

Imports the `test.v` file into the Virtuoso Studio 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 Studio 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 Studio 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`.

Example 6

```
cdsTextTo5x -lib myLib file1.sv file2.sv
```

Imports the 5x structure with default cell names for multiple SystemVerilog files.

Example 7

```
cdsTextTo5x -lib myLib file1.sv file2.v
```

Imports the files created in different languages based on the file extension.

Example 8

```
cdsTextTo5x -lib myLib -LANG verilog file1.sv file2.v
```

Imports the files in Verilog format irrespective of file extension.

Related Topics

Importing a Verilog File Using the cdsTextTo5x Command

Design Data Import by Using cdsTextTo5x

Benefits of cdsTextTo5x

Comparison between cdsTextTo5x and Virtuoso Text Editor

Performance of cdsTextTo5x

3

Design Data Import 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 tool 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

Related Topics

Syntax Rules for Text Views

Customization of Text Cellview by Using SKILL Flags

Virtuoso Text Editor User Guide

Syntax Rules for Text Views

The following syntax rules apply to analog language statements in text views.

Language	Line continuation (End of current line)	Line continuation (Beginning of next line)	Comment (Beginning of line)
Spectre	\\	(" ")	//
DSPF	""	+	**
PSPICE	\\	(" ")	*
SPICE	""	+	*
HSPICE	""	+	*
CDL	""	+	*

Related Topics

Customization of Text Cellview by Using SKILL Flags

Design Data Import by Using Virtuoso Text Editor

Customization of Text Cellview by Using SKILL Flags

You can use flags to customize the operation of text cellviews. Specify these flags either in the `.cdsinit` file or in the CIW.

The flags 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`

If the character count in port list declaration line exceeds 80 characters, the line is broken with the line continuation character '+' added at the end.

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

VHDL Flags

vhdlCreateEntityFromArch
vhdlCrossViewCheck
vhdlKeepCaseAsNC
vhdlIdentCaseSensitive
vhdlUpdateSymbol

Verilog Mixed-Signal Flags

vmsAnalysisType
vmsCreateMissingMasters
vmsCrossViewCheck
vmsDoNotCheckMasterFileWritable
vmsEnableAnsiHeader
vmsEnableImpInhConn
vmsEnableRapidImport
vmsNcvlogExecutable
vmsPortProcessing
vmsRunningInUI
vmsSensitivityIssuesReport
vmsTemplateScript
vmsTemplateScriptForSystemVerilog
vmsTemplateScriptForVerilog
vmsUpdatePcdb
vmsUpdateSymbolAfterEdit
vmsUserDefinedLibList
vmsVerboseMsgLevel

vhdlCreateEntityFromArch

```
vhdlCreateEntityFromArch = { t | nil }
```

Description

When set to `t`, creates an entity view for VHDL views from the architecture when a symbol view is present. The architecture view may be extracted from Virtuoso Text Editor or external editors. When set to `nil`, disables creation of entity views.

Example

```
vhdlCreateEntityFromArch = nil
```

Related Topics

vhdlCrossViewCheck

```
vhdlCrossViewCheck = { t | nil }
```

Description

When set to `t`, checks symbol views for consistency with other views in a cell when the `vmsUpdateCellViews` function is run. When set to `nil`, consistency checks are disabled.

Example

```
vhdlCrossViewCheck = nil
```

Related Topics

vhdlKeepCaseAsNC

```
vhdlKeepCaseAsNC = { t | nil }
```

Description

By default, names of VHDL identifiers (such as entity, port and architecture names) are lower cased when the vmsUpdateCellViews function is run.

When set to `t`, preserves the case of entity and port names when the `vmsUpdateCellViews` function is run. For escaped names, the case is preserved for all identifiers. You must also add the following entry in the hdl.var file:

```
define ncvhdlopts -keepcase4use5x
```

When set to `nil`, names of VHDL identifiers are lowercased.

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 is not modified. Instead, a new VHDL entity view is created.

Example

```
vhdlKeepCaseAsNC = t
```

vhdlIdentCaseSensitive

```
vhdlIdentCaseSensitive = { t | nil}
```

Description

Specifies whether names of objects, such as, pins, signals, aliases, and instance names are case sensitive. The default value is `nil`, which indicates that object names are not case sensitive.

Example

```
vhdlIdentCaseSensitive = t
```

vhdlUpdateSymbol

```
vhdlUpdateSymbol = { t | nil | query }
```

Description

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

Possible values are:

- `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.

Example

```
vhdlUpdateSymbol = t
vhdlUpdateSymbol = query
```

vmsAnalysisType

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

Description

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

Possible values are:

- `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.

Example

```
vmsAnalysisType = "Digital"
vmsAnalysisType = "Mixed"
```

vmsCreateMissingMasters

```
vmsCreateMissingMasters = { t | nil }
```

Description

When set to 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.

The default value is `nil`, which indicates that the environment does not create skeleton descriptions or symbols for undefined modules.

Example

```
vmsCreateMissingMasters = t
```

vmsCrossViewCheck

```
vmsCrossViewCheck = { t | nil }
```

Description

The default is `t`, which indicates that symbol views are checked for consistency with other views in a cell when the `vmsUpdateCellViews` function is run. When set to `nil`, consistency checks are disabled.

Example

```
vmsCrossViewCheck = nil
```

vmsDoNotCheckMasterFileWritable

```
vmsDoNotCheckMasterFileWritable = { t | nil }
```

Description

Checks the master file for write privileges and updates cellviews using the `vmsUpdateCellViews` function when the source file is read-only. The default value is `nil`, which indicates that the `vmsUpdateCellViews` function will not perform the update if the master file is read-only.

Example

```
vmsDoNotCheckMasterFileWritable = t
```

vmsEnableAnsiHeader

```
vmsEnableAnsiHeader = { t | nil }
```

Description

Prints the module header in the ANSI style of port declarations. Explicitly named ports are not supported in ANSI style port declarations, therefore, a non-ANSI style header is printed for such ports. The default value is `nil`, which indicates that the port declarations are printed in non-ANSI style.

Example

```
vmsEnableAnsiHeader = t
```

vmsEnableImpInhConn

```
vmsEnableImpInhConn = { t | nil }
```

Description

Prints inherited connections and `inout` directions in the port list of the behavioral view with the prefix `inh_` when explicit or implicit inherited connections are present in the schematic view of a cell at any level of the cell hierarchy. The name of the behavioral view must be `behavioral`. This is applicable for both Verilog behavioral and SystemVerilog text views.

When both `vmsEnableImpInhConn` and `hnlInhConnUseDefSigName` are set to `t`, the default signal name for terminals names is printed in the port list of the behavioral view instead of the prefixed property name.

Example

```
vmsEnableImpInhConn = t
```

vmsEnableRapidImport

```
vmsEnableRapidImport = { t | nil }
```

Description

When the *Check and Save* command is invoked, enables rapid import of large netlists by using the Text Infrastructure framework, which is also used by the `cdsTextTo5x` binary. Using this framework reduces the capacity limitations encountered while importing netlists for large designs. This functionality is currently supported in Verilog, Verilog-AMS and SystemVerilog text views.

The default value is `t`, which means that the Text Infrastructure framework is used to import netlists when *Check and Save* is invoked.

Note: The features supported by the environment variable `"ams.netlisterOpts" "amsTempDirForShadows"` and the SKILL flag `vmsCreateMissingMasters` are not supported when `vmsEnableRapidImport` is set to `t`.

Example

```
vmsEnableRapidImport = t
```

vmsNcvlogExecutable

```
vmsNcvlogExecutable = "path_and_executable"
```

Description

Specifies the name and path of the `ncvlog` executable that must be used to parse the Verilog-AMS text file.

Example

```
vmsNcvlogExecutable = "path_and_executable"
```

vmsPortProcessing

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

Description

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

Possible values are:

- `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
```

vmsRunningInUI

```
vmsRunningInUI = { t | nil }
```

Description

Displays messages in dialog boxes rather than in the CIW. When set to `nil`, messages are displayed in the CIW.

Example

```
vmsRunningInUI = t
```

vmsSensitivityIssuesReport

```
vmsSensitivityIssuesReport = "Summary" | "Detailed" | "Ignore"
```

Description

Controls whether the tool reports the issues that occur when the sensitivity properties are applied to a terminal during text extraction.

Possible values are:

- `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.

Example

```
vmsSensitivityIssuesReport = Detailed
```

```
vmsSensitivityIssuesReport = Ignore
```

vmsTemplateScript

```
vmsTemplateScript = "template_script" | nil
```

Description

Generates customized header information for new Verilog-AMS cellviews using the specified name of a script. The default is `nil`, which indicates that a default header is used in the following format:

```
//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
```

vmsTemplateScriptForSystemVerilog

```
vmsTemplateScriptForSystemVerilog = "template_script" | nil
```

Description

Generates customized header information for new SystemVerilog cellviews using the specified name of a script. The default is `nil`, which indicates that a default header is used in the following format:

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

For more information, see the example in vmsTemplateScript.

Example

```
vmsTemplateScriptForSystemVerilog = "template_script"
```

vmsTemplateScriptForVerilog

```
vmsTemplateScriptForVerilog = "template_script" | nil
```

Description

Specifies the name of a script used to customize the header information for new Verilog cellviews. When set to `nil`, a default header is used in the following format:

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

For more information, see the example in vmsTemplateScript.

Example

```
vmsTemplateScriptForVerilog = "template_script"
```

vmsUpdatePcdb

```
vmsUpdatePcdb = { t | nil }
```

Description

Updates the `pc.db` file of the source cellview automatically while doing a cellview-to-cellview operation. The default is `nil`, which indicates that the `pc.db` file of the source cellview is not updated automatically during a cellview-to-cellview operation.

Possible values are:

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

Example

```
vmsUpdatePcdb = t
```

vmsUpdateSymbolAfterEdit

```
vmsUpdateSymbolAfterEdit = { t | nil | query }
```

Description

Creates symbol views automatically for cells that do not have a symbol view when the vmsUpdateCellViews function is run.

Possible values are:

- `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.

Example

```
vmsUpdateSymbolAfterEdit = t
vmsUpdateSymbolAfterEdit = nil
```

vmsUserDefinedLibList

```
vmsUserDefinedLibList = l_library
```

Description

Specifies a list of libraries so that the search duration for the switch master is limited to the specified libraries.

If a list of libraries is not specified, an attempt is made to locate the symbol or text view in the current working directory. If these views are not found, another attempt is made to locate the symbol or text view in all directories that are defined in the `cds.lib` file. The SKILL function `ddGetLibList()` can be used to retrieve this information from the `cds.lib` file.

If a list of libraries is specified, an attempt is made to locate the symbol or text views in the directories that are defined in this list, but each library defined in this list should also be defined in the `cds.lib` file.

Example

```
vmsUserDefinedLibList = l_library
```

vmsVerboseMsgLevel

```
vmsVerboseMsgLevel = message_level
```

Description

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

Example

```
vmsVerboseMsgLevel = message_level
```

```
\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.
```