

VHDL In User Guide

Product Version IC23.1
June 2023

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

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

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

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

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

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

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

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

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

Contents

1

<u>Introducing VHDL In</u>	9
<u>Licensing Requirements</u>	10
<u>Uses of VHDL In</u>	11
<u>Other Cadence Products Used with VHDL In</u>	11
<u>The ncvhdl Parser</u>	12
<u>Design Flow Using VHDL In</u>	12
<u>VHDL In Software Directory</u>	13
<u>Starting Options and Requirements</u>	13
<u>Starting Options</u>	13
<u>Input Requirements</u>	14
<u>Conversion Limitations</u>	14

1

<u>Getting Started with VHDL In</u>	15
<u>Introduction</u>	16
<u>Setting Up the Library Environment</u>	16
<u>Setting Up the cds.lib File</u>	16
<u>Using the Text Editor</u>	17
<u>Using the CIW</u>	17
<u>Starting VHDL In in Standalone Mode</u>	22
<u>Starting VHDL In with Virtuoso Studio Design Environment</u>	22
<u>CIW Menu Options</u>	23
<u>Using SKILL APIs</u>	26

1

<u>VHDL In Forms</u>	27
<u>VHDL Import Form</u>	27
<u>Import Options</u>	29
<u>Power</u>	31

<u>Ground</u>	32
<u>Schematic Generation Options Tab Page</u>	32

1

<u>Importing a Simple VHDL Design</u>	39
<u>Introduction to the Example Design</u>	40
<u>Checklist Before Importing a Design</u>	40
<u>Setting Up the Library Environment</u>	41
<u>Importing to VHDL Cells</u>	41
<u>Setting Up the VHDL Import Form</u>	41
<u>Creating a New Target Library</u>	44
<u>Viewing the Results</u>	44
<u>Importing to a Netlist</u>	48
<u>Setting Up the VHDL Import Form</u>	48
<u>Creating a New Target Library</u>	49
<u>Viewing the Results</u>	50
<u>Importing to a Schematic</u>	54
<u>Setting Up the VHDL Import Form</u>	54
<u>Creating a New Target Library</u>	55
<u>Viewing the Results</u>	55

1

<u>Importing a Complex VHDL Design (RISC Processor Unit)</u> ..	61
<u>Overview</u>	62
<u>RPU Design Structure</u>	62
<u>Setting Up the VHDL Import Form</u>	63
<u>Specifying Schematic Generation Options</u>	65
<u>Creating a Target Library</u>	67
<u>Viewing Error and Log Files</u>	67
<u>Viewing the Results</u>	68
<u>Accessing the Symbol Editor</u>	68
<u>Viewing the Schematic</u>	70

1

<u>Conversion Issues</u>	77
<u>Introduction</u>	78
<u>Binding Issues</u>	78
<u>One Component, Multiple Entity/Architecture Pairs</u>	78
<u>One Entity, Multiple Components</u>	79
<u>Case Sensitivity Issues</u>	80
<u>Possible VHDL Design Import Errors</u>	80
<u>Object Search in OA</u>	81
<u>Other Issues</u>	83
<u>Signal Declarations</u>	83
<u>Vector Nets</u>	84
<u>Noninteger Vector Indices</u>	84
<u>Buffer Ports</u>	85
<u>Port Conversion Functions</u>	85
<u>Attribute Specifications</u>	86
<u>Importing Generics</u>	86
<u>Component Declarations</u>	87
<u>Port Maps</u>	88
<u>Concurrent Assignment Statements</u>	88
<u>Behavioral View for Continuous Signal Assignment Statements</u>	89
<u>Power and Ground Signals</u>	89
<u>Nontranslatable Structural VHDL Constructs</u>	90
<u>Port Subtypes and Modes</u>	90
<u>Constructs in Architectures</u>	92

1

<u>Error Messages</u>	95
<u>Dialog Boxes</u>	96
<u>File Name Does Not Exist</u>	96
<u>No Target Library Name Specified</u>	96
<u>Library Name Not Found</u>	96
<u>Import Completed Successfully</u>	97
<u>Import Completed with Errors/Warnings</u>	97

VHDL In for Virtuoso Design Environment User Guide and Reference

<u>Import Aborted When Reached Maximum Errors</u>	97
<u>Text String Error Messages</u>	98
<u>Overview</u>	98
<u>Line Command Errors</u>	98
<u>Analyzer Errors</u>	99
<u>Parameter File Parsing</u>	99
<u>Virtuoso Design Library</u>	99
<u>Memory</u>	100
<u>General Errors</u>	100
<u>Using the VHDL Tool Box Status Window</u>	100

1

<u>Creating a Parameter File</u>	103
<u>Parameter File Structure</u>	104
<u>Description of Parameters</u>	104

1

<u>VHDL In Standalone Options</u>	113
<u>Introduction</u>	114
<u>Getting Help on a VHDL In Command</u>	115
<u>Displaying the Version Number of VHDL In</u>	116
<u>Suppressing Printing of the Copyright Banner</u>	116
<u>Hiding the Display of Schematic Extraction Errors</u>	116
<u>Specifying a cds.lib File</u>	117
<u>Compiling Functional Cellviews</u>	117
<u>Specifying the VHDL WORK Library</u>	117
<u>Ignore Extra Pins on Symbols</u>	118
<u>Speeding Up Run Time</u>	118
<u>Specifying a Schematic Parameter File</u>	119
<u>Specifying Multiple VHDL Source Files</u>	119
<u>TDM and Imported VHDL Design Libraries</u>	120
<u>VHDL In operation in TDM Mode</u>	120
<u>Checking a Library into TDM</u>	120
<u>Enabling VHDL93 features</u>	121

<u>Glossary</u>	123
-----------------------	-----

Introducing VHDL In

VHDL In lets you import the designs written in the VHDL language to Virtuoso database format. The import process converts the textual designs into the schematic, netlist, or textual form, depending on the options you set.

This user guide describes how to use VHDL In. It is aimed at the designers of digital circuits and assumes that you are familiar with:

- The Virtuoso Studio design environment and application infrastructure mechanisms designed to support consistent operations between all Cadence tools.
- The applications used to design and develop integrated circuits in the Virtuoso Studio design environment, notably Virtuoso Schematic Editor.
- Virtuoso technology data.
- Designing and simulating electronic designs based on VHDL with Cadence design and simulation software.

This chapter discusses the following:

- Licensing Requirements
- Uses of VHDL In
- Other Cadence Products Used with VHDL In
- Design Flow Using VHDL In
- VHDL In Software Directory
- Starting Options and Requirements
- Conversion Limitations

Licensing Requirements

VHDL In searches for the following licenses in the specified order and checks out one of them:

- 1 license of Virtuoso® Schematic Editor L
- 1 license of Virtuoso® Schematic Editor XL
- 1 license of Virtuoso® Layout Suite L
- 1 license of Virtuoso® Layout Suite XL
- 4 tokens of Virtuoso® Layout Suite GXL

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

Uses of VHDL In

When you use VHDL In for Virtuoso® Design Environment, you can 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 views

In every case, VHDL In imports the design from the VHDL format into a Virtuoso Studio Design Environment database format—a data format that can be used by Cadence tools.

You can import the following into a Virtuoso® Design Environment library:

- A VHDL design
- A VHDL ASIC library
- A VHDL design, minus modules that already exist in the Virtuoso® Design Environment library
- Pieces of a hierarchical design
- A combination of the above architecture

If you convert your VHDL design into schematics, you can edit the schematics with Virtuoso Schematic Editor L.

Other Cadence Products Used with VHDL In

VHDL In is available as part of the VHDL Interface product or by itself. VHDL Interface in Virtuoso® Design Environment is available as part of the VHDL Virtuoso Schematic Editor L software package, or in combination with only the Schematic Generator.

VHDL In is also available with the Virtuoso Schematic Editor L and Synergy™ products. If you do not have Virtuoso Schematic Editor L and Synergy to generate schematics, you can buy the Composer Schematic Generator product, which includes the VHDL In and Verilog In™ tools.

VHDL In requires a parser to parse the designs before these are created in OA database. The ncvhdl parser is a default parser for parsing the designs.

The ncvhdl Parser

The ncvhdl parser and other libraries, such as IEEE and STD required to parse the designs, are available in both hierarchies, IUS and dfll. However, IUS hierarchy contains the latest version of the ncvhdl parser and libraries. These are used from IUS hierarchy if either of the following IUS versions is available:

- 05.83-s009 or higher in IUS 05 series
- 06.11-s004 or higher in IUS 06 series

If none of these versions is available, the parser and libraries are used from the dfll hierarchy. To use the ncvhdl parser in the standalone mode, start VHDL In using the following command:

```
vhdlin
```

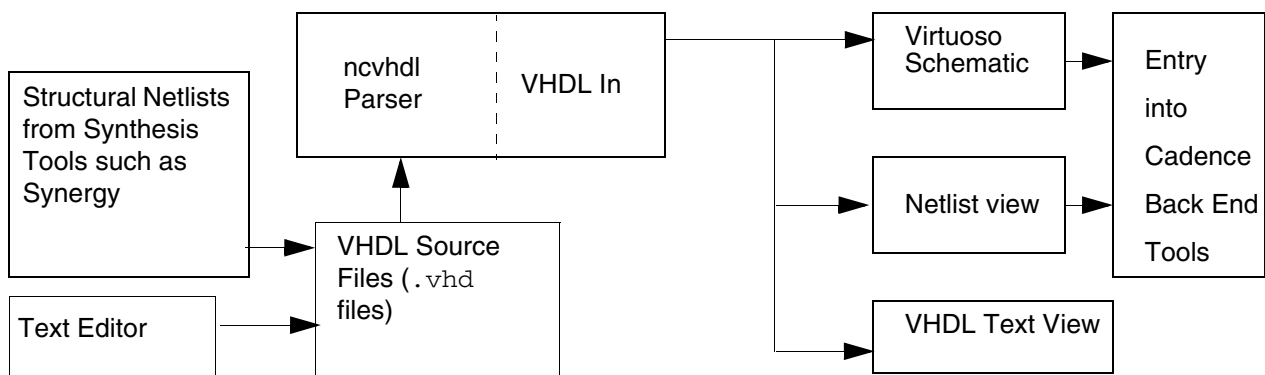


Starting IC 6.1.4, VHDL In does not support usage of the Leapfrog parser. Only ncvhdl parser is used.

Design Flow Using VHDL In

The following diagram shows how you use VHDL In as part of the VHDL design process.

VHDL In Design Flow



The .oa file generated by VHDL In is in OA database format, the design data storage format used by Cadence tools.

VHDL In Software Directory

The VHDL In software is located in

```
<dfII_install_dir>/tools/dfII/bin/vhdlin
```

where

<dfII_install_dir> is a variable signifying the directory where your Cadence software is installed.

To find out where the Cadence software is installed, enter `cds_root` at the UNIX command prompt.

`cds_root` is a utility that identifies the location of Cadence installation hierarchies. Startup scripts typically use it to find the location of Cadence installation hierarchies before starting tools.

`cds_root` requires one argument. This is an executable name.

Use the following syntax for `cds_root`:

```
cds_root executableName
```

Here, `executableName` is either an executable found in `$PATH` or is a full path name to an executable.

`cds_root` uses the `executableName` argument to identify the installation hierarchy of the tool and to check if it is a legal hierarchy. If you specify the full path to the executable, then `cds_root` checks only that location to see if it is a legal hierarchy.

If the executable is not found in `$PATH` or is not located in the hierarchy, `cds_root` displays the following error message:

```
Error! Cant determine installation root from PATH
```

Starting Options and Requirements

There are two ways to set up and run VHDL In. You must have several items ready before you start.

Starting Options

You can start VHDL In in two ways:

- From the menu in the Virtuoso Studio Design Environment Command Interpreter Window (CIW).
- In the standalone mode Chapter 1, “Getting Started with VHDL In,” describes these options in more detail.

Input Requirements

Before starting VHDL In in Virtuoso Studio Design Environment, you need these input elements:

- The VHDL design files
- The correct library environment

To start VHDL In in standalone mode, you need

- A file listing the parameters
- VHDL design files

You can place the file names in a file, or enter them directly at the command prompt.

Conversion Limitations

Due to some fundamental differences between the text files used in VHDL and the structural component files used in schematics and layout, VHDL In cannot convert some elements of IEEE standard 1076 VHDL into Virtuoso schematics or OA format netlists. VHDL In can convert some elements of a VHDL design, but only after you modify them. VHDL In makes certain assumptions when it converts VHDL elements into Cadence format.

Chapter 1, “Conversion Issues,” details the constructs in VHDL that can or cannot be converted, as well as the limitations and by-products of the conversion process.

Getting Started with VHDL In

This chapter discusses the following:

- [Introduction](#)
- [Setting Up the Library Environment](#)
- [Starting VHDL In in Standalone Mode](#)
- [Starting VHDL In with Virtuoso Studio Design Environment](#)
- [Using SKILL APIs](#)

Introduction

You can use VHDL In in the following modes:

- Interactive mode

You can launch the graphical user interface of VHDL In from the Command Interpreter Window ([CIW](#))

- Standalone mode

You can use VHDL In from the command prompt

Either way, you must set up your library environment before you can use VHDL In.

Setting Up the Library Environment

To set up the library environment required for VHDL In, you install the latest Cadence software in the installation directory and set up a `cds.lib` file. The `cds.lib` file includes the libraries you need to import designs.

Setting Up the `cds.lib` File

Create the `cds.lib` file in your home directory or the directory in which you run Cadence software.

In the `cds.lib` file, you include a list of paths to

- The five required reference libraries: `basic`, `sample`, `US_8ths`, `std`, and `ieee`
- Additional libraries you need

You can add the required reference libraries and additional libraries you need by using a text editor or by using the Virtuoso® Design Environment CIW.

You can create a new library in the CIW without creating a `cds.lib` file beforehand. In this case, the Library Manager tool creates a `cds.lib` file in the directory where you started VHDL In. The Library Manager tool creates additional entries in the `cds.lib` file for each library or design that you add or create in the CIW.

The syntax of entries for required and additional libraries in the `cds.lib` file is the same whether the entries are created by you or by the Library Manager tool. The syntax for an entry is


```
DEFINE library_name absolute_path/library_name
```

Using the Text Editor

You can use a text editor in an xterm window to

- Create a `cds.lib` file
- Add the required reference libraries to a `cds.lib` file
- Add your own libraries to a `cds.lib` file

Be sure to enter the paths to the libraries in the same case as they occur, so that these libraries can be found.

1. Use your text editor to create the `cds.lib` file and open it for editing.

This example shows the command for the vi editor.

```
vi cds.lib
```

2. Include the following set of reference libraries in your `cds.lib` file. (Use uppercase and lowercase characters precisely as you see here.)

`<dfII_install_dir>` is a variable signifying the particular path you use for the Cadence installation directory.

```
DEFINE basic <dfII_install_dir>/tools/dfII/etc/cdslib/basic
DEFINE sample <dfII_install_dir>/tools/dfII/samples/cdslib/sample
DEFINE US_8ths <dfII_install_dir>/tools/dfII/etc/cdslib/sheets/US_8ths
DEFINE std <IUS_install_dir>/tools/inca/files/STD
DEFINE ieee<IUS_install_dir>/tools/inca/files/IEEE
```

3. Add additional libraries you want to use.

Example entries in a `cds.lib` file for additional design libraries are

```
DEFINE adder /usr/designer/data/4.4.1/adder
DEFINE mixed /usr/designer/data/4.4.1/mixed
DEFINE msheet /usr/designer/data/4.4.1/msheet
```

4. Write and quit the file.

Using the CIW

You can use the CIW to create a `cds.lib` file and add libraries to the file. Use the Library Path Editor form (labeled *cdsLibEditor*) to add the required reference libraries. You can use either the Library Path Editor form or the New Library form to add your own libraries to the `cds.lib` file.

If you create a new library in the New Library form without creating a `cds.lib` file beforehand, the Library Manager tool creates a `cds.lib` file in the directory where you started VHDL In. The Library Manager tool creates entries in the `cds.lib` file for each library or design that you add or create in the CIW.

If you use other Cadence applications (and have more than one `cds.lib` file) in addition to VHDL In, be sure that you start Virtuoso® Design Environment from a directory that contains the `cds.lib` file you want to use for VHDL In.

Adding the Required Reference Libraries or Additional Libraries

Use the Library Path Editor form in the CIW to:

- Add the five required reference libraries (`basic`, `sample`, `US_8ths`, `std`, and `ieee`) to the `cds.lib` file.
- Specify new libraries for designs you create.
- Edit the names and paths to libraries already in the your `cds.lib` file.

To use the Library Path Editor form:

1. Enter in an xterm window

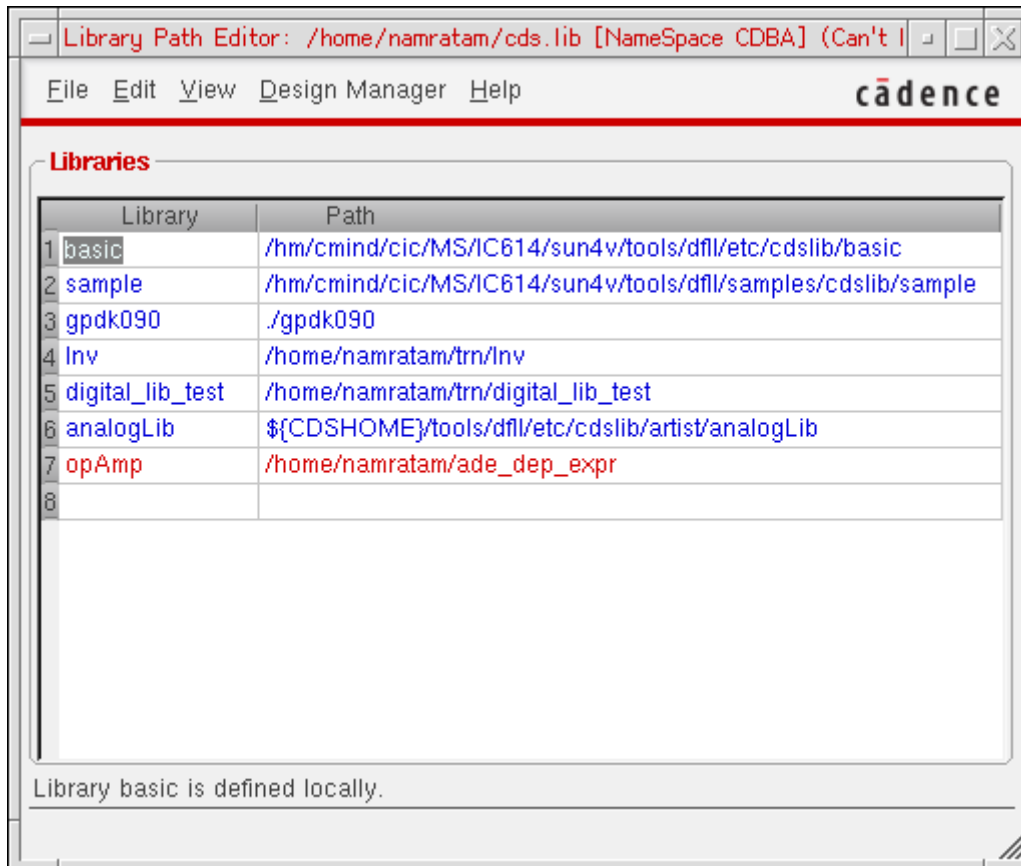
```
virtuoso &
```

The CIW appears.

2. In the CIW, select *Tools – Library Path Editor*

The Library Path Editor form appears.

This form displays any libraries currently in your `cds.lib` file.



3. In the Library and Path text fields, enter the library name and path for the required reference libraries, or for additional libraries you want to use.

Use lowercase for the IEEE library name in the *Library* field, but use uppercase for the IEEE library name in the *Path* field. This way, you accommodate the name mapping that the parser performs on the reference libraries when it parses a design you are importing.

4. Select *File – Exit*.

Adding Only Additional Libraries

You can use the New Library form (in addition to the Library Path Editor form) to add additional design libraries to the `cds.lib` file.

1. Enter in an xterm window

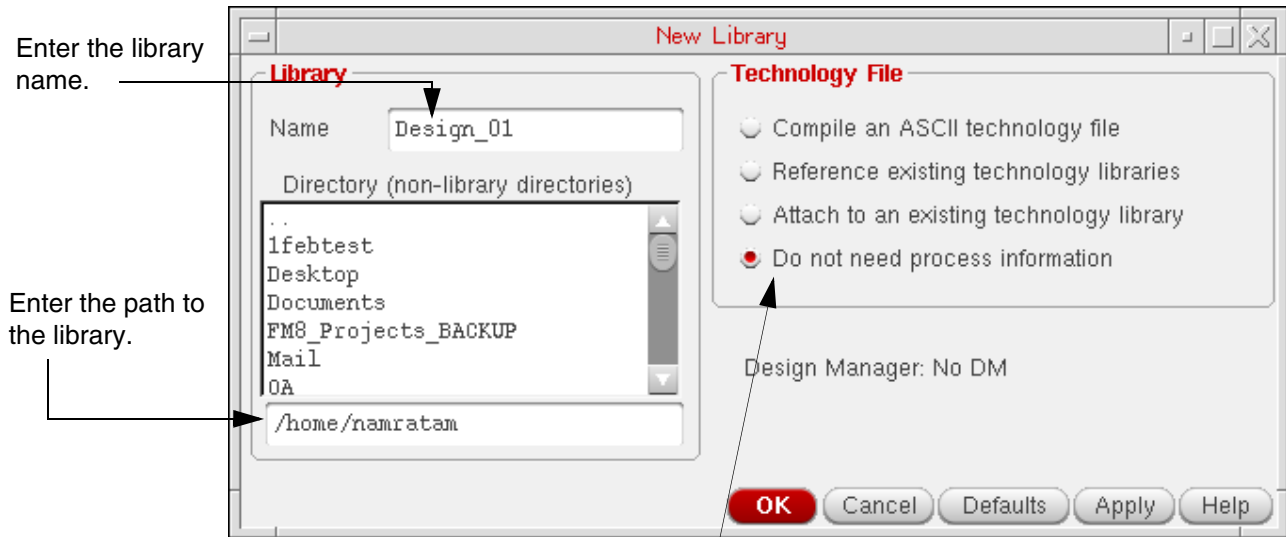
```
virtuoso &
```

The CIW appears.

2. In the CIW, select *File – New – Library*

The New Library form opens.

3. Enter the name of the new library and the path to its location



You can use the scroll bar in the directory window to fill in the library path field. Click a directory name to enter it into the path field.

4. In the *Name* field, specify the library name.

`Design_01`

5. In the field below the *Directory* list box, enter the path to the new library.

`<dfII_install_dir>/Design_01`

`<dfII_install_dir>` is a variable signifying the particular path you use for the Cadence installation directory.

6. Click *Do not need process information*.

Because the library does not contain layout data, you do not need a technology file.

7. Click *Apply*.

The form stays open, so that you can create another library.

When you create the last new library, click *OK* instead of *Apply*, to close the New Library form.

The new library is created.

8. In the CIW, select *Tools – Library Manager*.

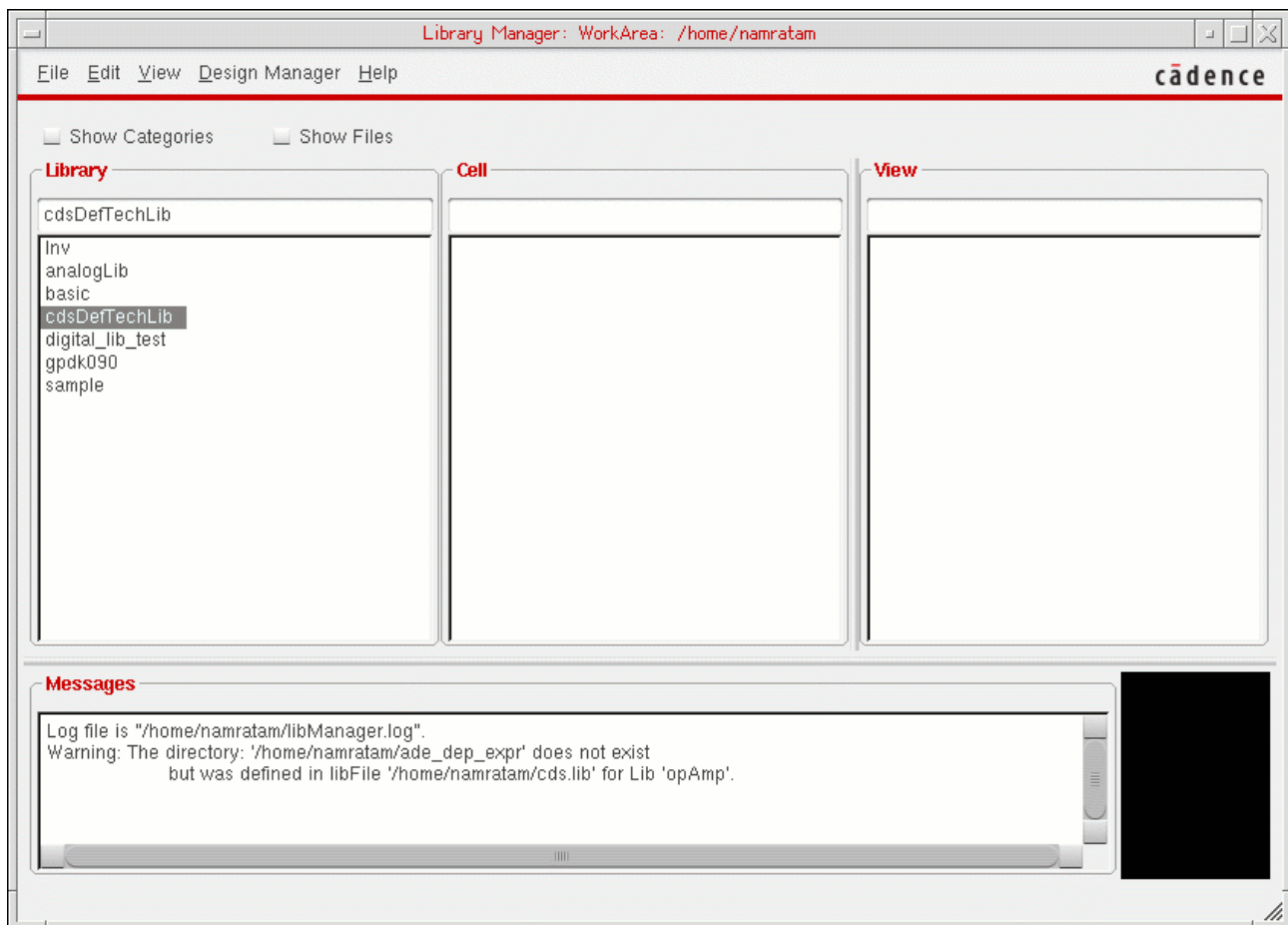
The Library Manager window opens, displaying the name of the library you just created.

9. Click the name of the new library.

The contents of the new library (*ieee*) appear in the *Cell* column.

In the following example, the *Show Categories* option is turned off and you can see references to the five required libraries (*basic*, *sample*, *US_8ths*, *std*, and *ieee*) and user-defined library *cdsDefTechLib*.

The contents of the library you selected appear in the *Cell* column. If problems occur during the creation or addition of the new library, error and warning messages appear in the *Messages* window.



Note: If the basic library does not contain a patch cod symbol, or the basic library is not defined in `cds.lib`, VHDL In fails to create a schematic. It creates a behavioral view instead of a schematic.

Starting VHDL In in Standalone Mode

If you do not have Virtuoso® Design Environment running (no CIW is displayed), you can run VHDL In in standalone mode.

- In a terminal window, type the following command:

```
vhdlin -param pfn -f fn <vfn1, vfn2, ...>
```

where

the `pfn` argument to the `-param` option indicates the name of the file that contains parameters. The contents of this file are described in detail in [Chapter 8, “Creating a Parameter File.”](#)

The `fn` argument to the `-f` option indicates the path and file name of the file that contains the names of the VHDL design unit files

```
<vfn1, vfn2, ...>
```

is an optional list of the individual VHDL files.

Starting VHDL In with Virtuoso Studio Design Environment

To start VHDL In with Virtuoso Studio Design Environment

- Enter in an xterm window

```
virtuoso &
```

The Command Interpreter Window (CIW) for Virtuoso Studio Design Environment appears.

To start the process of importing a design, open the VHDL Import form. You can access this form either through menu options in the CIW, or with a SKILL command at the command prompt in the CIW.

CIW Menu Options

You can use either of two sequences of menu options in the CIW to access the VHDL Import form:

- *File – Import – VHDL*
- *Tools – VHDL Toolbox*

File – Import – VHDL

To access VHDL In through the CIW.

- ➔ On the CIW, choose *File – Import – VHDL*.

The VHDL Import form opens.

VHDL In for Virtuoso Design Environment User Guide and Reference

Getting Started with VHDL In

VHDL Import

Import Options | Schematic Generation Options

File Name:

Target Library Name:

File List:

- ./
- .Trash/
- .VirtuosoRDE/
- .acrobat/
- .adobe/
- .artist_states/
- .cadence/

Buttons: Add >> << Remove

Path: /home/namratam/*.vhd

Import Structural Architectures As: schematic

Reference Libraries: basic US_8ths ieee std sample

Symbol View Name: symbol

Overwrite Existing Views: ☒

Overwrite Symbol Views: None

Case Sensitive Symbol Matching: ☒

User Specified Standard Libraries: ☐

Maximum Number of Errors: 10

Compile VHDL Views After Import: ☐

Compiler Options:

VHDL WORK Library Name:

Summary File: ./vhdlin.summary

Compatibility Option: ☐

v93 Option: ☐

Power

Net Name: vdd! Value: '1'

Data Type: std_ulogic

Ground

Net Name: gnd! Value: '0'

Data Type: std_ulogic

Buttons: OK Cancel Defaults Apply Help

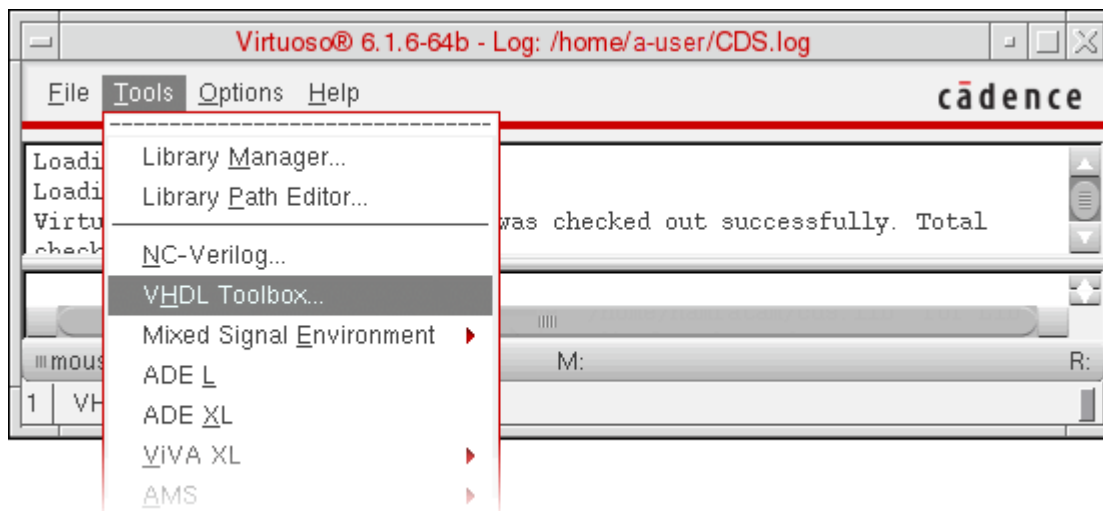
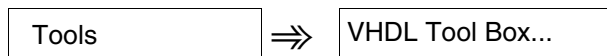
The *Files List* Box shows only files with a `.vhd` suffix and all directories in the local directory.

All the different fields on the form are explained in [Chapter 3, “The VHDL Import Form.”](#)

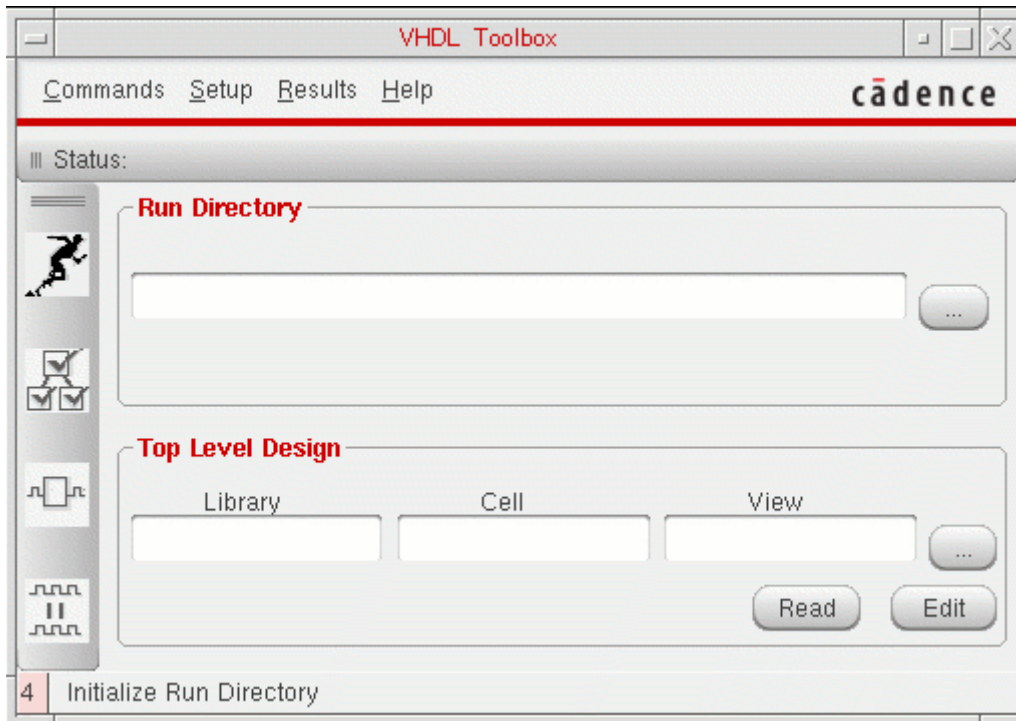
Tools – VHDL Tool Box

To open VHDL Import from VHDL Toolbox:

1. On the CIW, choose *Tools – VHDL Tool Box*.



The VHDL Toolbox form appears.



2. Choose *Commands – Import*.

The VHDL Import form appears.

Using SKILL APIs

You can use various SKILL APIs to operate VHDL In. using SKILL APIs, you can perform the following operations:

- Build and display the VHDL Import form.
- Import a list of VHDL source files into the specified library with the given parameters.
- Translate a VHDL cellview into an intermediate pin list format.
- Generate VHDL views from an intermediate pin list format.
- Define SKILL procedures and register this information with the VHDL Toolbox using a SKILL routine.

For information on using SKILL functions of VHDL In, see *HDL Import and Netlist-to-Schematic Conversion SKILL Reference*.

VHDL In Forms

This chapter discusses the following:

- [VHDL Import Form](#)
- [Schematic Generation Options Tab Page](#)

VHDL Import Form

You can use the VHDL Import form to select VHDL files to import into a library and also specify the options used for importing.

VHDL In for Virtuoso Design Environment User Guide and Reference

VHDL In Forms

VHDL Import

Import Options | Schematic Generation Options

File Name:
Target Library Name:

Add >> << Remove
/home/namratam/*.vhd

Import Structural Architectures As: schematic
Reference Libraries: basic US_8ths ieee std sample
Symbol View Name: symbol
Overwrite Existing Views: ☒
Overwrite Symbol Views: None
Case Sensitive Symbol Matching: ☒
User Specified Standard Libraries: ☐
Maximum Number of Errors: 10
Compile VHDL Views After Import: ☐
Compiler Options:
VHDL WORK Library Name:
Summary File: ./vhdlin.summary
Compatibility Option: ☐
v93 Option: ☐

Power
Net Name: vdd! Value: '1'
Data Type: std_ulogic

Ground
Net Name: gnd! Value: '0'
Data Type: std_ulogic

OK Cancel Defaults Apply Help

The parameters corresponding to many options and fields in this form are described in [Chapter 1, “Creating a Parameter File.”](#)

File Name is the field where you enter the name of a source file to import, a directory path to select files from, or a file name pattern to match.

Files List box either lists all files that match the file name pattern or lists subdirectories located in the current directory.

Target Library Name is the field where you enter the name of the library where the results from the import process are to be stored.

Import Files List box lists the VHDL files chosen from the *Files List Box* field for importing.

Add adds selections from the *Files List Box* to the *Import Files List Box*.

Remove removes selections from the *Import Files List Box*.

Import Options

Import Structural Architectures As specifies the output format that VHDL In produces from the imported structural VHDL architectures. The default is *schematic*. The formats are

- *schematic* generates a Virtuoso Schematic Editor L schematic that graphically represents the VHDL architecture
- *netlist* generates an OA netlist that represents the connectivity of the VHDL architecture.
- *vhdl* imports the VHDL architecture as a text file to the target library.

This corresponds to the structuralViewType parameter in a parameter file.

Reference Libraries is a text field that you can use to specify the reference libraries for symbols when generating schematics or netlists for unbound or partially bound structural architectures. When the form first opens, this field contains the default library list *basic* and *US_8ths*. Separate additional library names from each other with a space (as the default libraries are).

This corresponds to the referenceLibraries parameter in a parameter file.

Symbol View Name specifies the view name to use for generating symbols during the import process. The default is *symbol*.

This corresponds to the symbolViewName parameter in a parameter file.

Overwrite Existing Views controls whether existing views in the target library are overwritten by the imported data. The checkbox is selected by default. The *vhdl* view is always overwritten.

This corresponds to the `overwriteExistingView` parameter in a parameter file.

Note: This option overwrites only the schematic, functional, or netlist views. It does not overwrite the symbol views. To overwrite symbol views, set the *Overwrite Symbol Views* option.

Overwrite Symbol Views controls whether existing symbols in the target library are overwritten by the imported symbols. By default, the option is set as `None` and the existing *symbol* views are not overwritten.

Set this option to any of the following values to specify which symbols you want to overwrite:

- **None** does not overwrite any existing symbol.
- **Created by VhdlIn** overwrites only those existing symbols that were created by VHDL In. These symbols have the `createdBy` property set as `vhdlIn`. Symbols created by the any other tool remain unaffected.



Tip

To view the `createdBy` property of any symbol, open it in *Virtuoso Symbol Editor L*. Choose the *Edit — Properties — Cellview* menu option. The Edit Cellview Properties form that is displayed shows the *createdBy* property field.

- **Created by TSG and Others** overwrites the existing symbols created by the Text-to-Symbol generator or any other tool. For these symbols, either the `createdBy` property is not set or it is set as any value other than `vhdlIn`.

For more details about the Text-to-Symbol generator, see the [Text To Symbol Generator Files](#) section.

- **All** overwrites the existing symbols with those imported by VHDL In.

This option corresponds to the `overwriteSymbolView` parameter in a parameter file.

Case Sensitive Symbol Matching specifies that symbol view name is case sensitive. By default, this option is selected.

User Specified Standard Libraries lets you specify your own standard libraries.

Note: You can use only CV compiled libraries. The path to the user libraries must be specified in `cds.lib`.

Maximum Number of Errors controls the maximum number of errors that can occur before the import process quits. The default is 10.

This corresponds to the `maxError` parameter in a parameter file.

Compile VHDL Views After Import is a box you click when you want your imported VHDL files to be parsed by the NC parser, `ncvhdl`. The option is deselected by default.

Compiler Options is the field where you specify the `ncvhdl` parser options you want to use when you set the Compile VHDL Views After Import to *on*. The default is an empty field.

VHDL WORK Library Name is the field where you specify the workarea that `ncvhdl` parser uses to store the intermediate results of the import process. You can use this work library when you want to import the design again.

This corresponds to the `work_file` parameter in a parameter file.

Summary File is the field where you enter the name to be given to the file that contains the combined contents of the log and the error files.

Compatibility option controls the enabling /disabling of the vendor compatibility mode. The default is OFF.

Note: Currently, the `ncvhdl` parser does not support compatibility option.

v93 option enables/disables the support of VHDL'93 features, while running the `ncvhdl` parser. The default is OFF.

Note: Enabling the *v93 option* will only make the parser to parse and pass VHDL'93 source files. VHDL In will import these files as `vhdl` views only.

Power

This option specifies the power net that VHDL In uses for power assignments in the schematic.

Net Name is the field where you specify the net name to assign to power. The default is `vdd!`. VHDL In automatically appends `!` if `!` is not present in the net name.

This corresponds to the `powerNetName` parameter in a parameter file.

Value is the field where you specify the value of the power signal (net) that is replaced by the net name you specify. The default is `1`.

This corresponds to the `powerLiterals` parameter in a parameter file.

Data Type is the field where you specify the VHDL data type for the power net. The default is `std_ulogic`.

This corresponds to the powerType parameter in a parameter file.

Ground

This option specifies the ground net that VHDL In uses for ground assignments in the schematic.

Net Name is the field where you specify the net name to assign to ground. The default is gnd! VHDL In automatically appends the "!" if "!" is not present in the net name.

This corresponds to the groundNetName parameter in a parameter file.

Value is the field where you specify the value of the ground signal (net) that is replaced by the net name you specify. The default is 0.

This corresponds to the groundLiterals parameter in a parameter file.

Data Type is the field where you specify the VHDL data type of the ground net. The default is *std_ulogic*.

This corresponds to the groundType parameter in a parameter file.

Schematic Generation Options Tab Page

Use the *Schematic Generation Options* tab to generate a schematic view of your design. The parameters corresponding to many options and fields in this form are described in Chapter 1, "Creating a Parameter File."

- On the VHDL Import form, click the *Schematic Generation Options* tab.

The *Schematic Generation Options* tab appears, as shown in the figure below.

VHDL In for Virtuoso Design Environment User Guide and Reference

VHDL In Forms

VHDL Import

Import Options | **Schematic Generation Options**

Sheet Border Size: none

Maximum Number of Rows: 1024

Maximum Number of Columns: 1024

Font Height: 0.0625

Line To Line Spacing: 0.2

Line To Component Spacing: 0.5

Component Density: 0 (Lowest to Highest)

Pin Placement

☒ Left and Right Sides ☐ All Sides ☐ Pin Placement File

Pin Placement File Name:

Text to Symbol Generator Files:

Full Place and Route: ☒ Optimize Wire Label Locations ☒

Generate Square Schematics: ☒ Extract Schematics ☒

Minimize Crossovers: ☒ Ignore Extra Pins On Symbol ☐

Fast Schematic Generation

Generate Fast Schematic: ☒

Instances Greater Than: 20000 Ports Greater Than: 5000

OK Cancel Defaults Apply Help

Sheet Border Size

This is a cyclic field that controls the size of the sheet on which the schematic is generated. If Sheet Border Size is *none*, then VHDL In generates a single sheet schematic. If the Sheet Border Size is *A*, *B*, *C*, *D*, *E*, or *F* and the system cannot fit the schematic onto one sheet, it creates a multisheet schematic. A multisheet schematic has one index sheet and many schematic sheets. The default is *none*.

This corresponds to the sheetBorderSize parameter in a parameter file.

Maximum Number of Rows

This is the maximum number of rows of components allowed on each generated schematic sheet. The system uses this option only if it creates a multisheet schematic. The value must be an integer from 1 to 1024. The default is 1024.

Because of the size of the components and the size of the sheet, VHDL In might not be able to place the maximum number of rows on a sheet. If you specify a value that is too large for the size of the instance symbols and the size of the sheet, VHDL In places as many rows as it can fit on each sheet up to the maximum of 1024.

This corresponds to the maxNoRows parameter in a parameter file.

Maximum Number of Columns

This is the maximum number of columns of components allowed on each generated schematic sheet. VHDL In uses this option only if it creates a multisheet schematic. The value must be an integer from 1 to 1024. The default is 1024.

Because of the size of the components and the size of the sheet, VHDL In might not be able to place the maximum number of columns on a sheet. If you specify a value that is too large for the size of the instance symbols and the size of the sheet, VHDL In places as many columns as it can fit on each sheet up to the maximum of 1024.

This corresponds to the maxNoCols parameter in a parameter file.

Font Height

This option controls the height of the font used for pin, wire, and instance labels. The value is in user units. Pin labels are scaled down to 75 percent of the specified size. The default is 0.0625.

This corresponds to the fontHeight parameter in a parameter file.

Line To Line Spacing

This option controls the spacing between two adjacent nets. The value is in user units. The value ranges from 0.0 to 1000000.0. The default is 0.2.

This corresponds to the `lineLineSpacing` parameter in a parameter file.

Line To Component Spacing

This option controls the spacing between a component instance and an adjacent net. The value is in user units. The value ranges from 0.0 to 1000000.0. The default is 0.5.

This corresponds to the `lineComponentSpacing` parameter in a parameter file.

Component Density

This option controls the number of components allowed on a single sheet in the case of schematics that instantiate sheet borders. The value ranges from 0 to 100. The higher the density value, the greater the number of components on one page. The default is 0.

This corresponds to the `componentDensity` parameter in a parameter file.

Pin Placement

This option controls whether pins are placed in the generated schematic on the left and right sides only, on all four sides, or specified on a per pin basis in a pin placement file. The default is Left and Right Sides.

Text to Symbol Generator Files

Specify a space-separated list of tsg files to be used by VHDL In to generate the symbols in the target library. VHDL In internally runs the Text-to-Symbol generator, a tool that reads the symbol descriptions given in the tsg files to create symbol views.

For more details about the Text-to-Symbol generator and the tsg files, see [Text-to-Symbol Generator](#) in Virtuoso Schematic Editor L User Guide.

Important

If the tsg files are given, the direction of pins specified by the `Pin Placement` option is ignored. Pins are placed on symbols as defined in the tsg files.

Full Place and Route

This option controls whether the generated schematic is fully placed and routed. Disabling this option causes VHDL In to generate a schematic that is connected by name only, which

improves schematic generation performance in both time and memory. This option is selected by default.

This corresponds to the fullPlaceRouteSchematic parameter in a parameter file.

Generate Square Schematics

This option controls whether disproportionately long columns of components are broken up into many columns. This option is selected by default.

This corresponds to the squareSchematics parameter in a parameter file.

Minimize Crossovers

This option controls the aesthetic quality of the generated schematic by minimizing net crossovers. This option is selected by default.

This corresponds to the minimizeCrossovers parameter in a parameter file.

Optimize Wire Label Locations

This option controls whether the location of wire labels is optimized. Disabling this option improves schematic generation performance in both time and memory, but might result in overlapping of names. This option is selected by default.

This corresponds to the optimizeLabels parameter in a parameter file.

Ignore Extra Pins on Symbol

This option controls the selection of symbols from the reference libraries. If this option is specified and VHDL In finds a reference symbol with the same name as specified in the VHDL design, the symbol will be picked up. The pins not referred will remain unconnected in the schematic.

Note: The symbol will be picked up even if all its pins are not used.

Fast Schematic Generation

The options in this group box enable fast generation of the schematic when the design being imported contains a large number of instances or ports.

Fast Schematic Generation

Generate Fast Schematic ☒

Instances Greater Than Ports Greater Than

The following table lists the fields in the *Fast Schematic Generation* group box, with their default values and parameters. You can use the parameters in the parameter file or `.cdsenv` of VHDL In. For details on the parameter file, see [Chapter 1, “Creating a Parameter File.”](#)

Field	Default Value	Parameter
<i>Generate Fast Schematic</i>	On	<code>generateFastSchematic</code>
<i>Instances Greater Than</i>	20000	<code>fastSchematicMaxInst</code>
<i>Ports Greater Than</i>	5000	<code>fastSchematicMaxPort</code>

To use this feature, select the *Generate Fast Schematic* check box and specify the number of instances and ports in their respective fields. If the number of instances or ports in the design exceeds the specified number, the tool generates a schematic in which the instances are placed in a two-dimensional array without any routing, and the connectivity of the nets is indicated by names.

Notes:

- To enable the fast schematic generation feature without considering the number of instances or ports in the designs being imported, select *Generate Fast Schematic* and type 0 in the *Instances Greater Than* and *Ports Greater Than* fields.
- If you disable the fast schematic generation feature, and the design has a large number of instances and ports, schematic generation can take significant time to place and route the instances and nets.

Importing a Simple VHDL Design

This chapter discusses the following:

- [Introduction to the Example Design](#)
- [Checklist Before Importing a Design](#)
- [Setting Up the Library Environment](#)
- [Importing to VHDL Cells](#)
- [Importing to a Netlist](#)
- [Importing to a Schematic](#)

Introduction to the Example Design

In this chapter, you import a simple example VHDL design. This example is included in the Cadence software hierarchy, at `<dfll_install_dir>/tools/dfII/samples/vhdlin/Test3`, where `<dfll_install_dir>` is your installation directory.

The design is a full adder that includes a VHDL design of a half adder and an OR gate. The half adder contains the behavioral information, while the full adder is only structural. The entities (*e*) and architectures (*a*) are split into separate files. The VHDL design directory includes

```
full_adder.a.vhd
full_adder.e.vhd
half_adder.a.vhd
half_adder.e.vhd
or_gate.a.vhd
or_gate.e.vhd
```

You can import the design into each of the three types of outputs:

- VHDL text view
- Netlist view
- Schematic view

Checklist Before Importing a Design

Before you use VHDL In to import a design, use the following checklist to review what you need:

1. What do you want to convert the VHDL design into?

- ☐ Schematic view
- ☐ Netlist view
- ☐ VHDL text view

2. Do you have all the design units you need?

- ☐ Entity declarations
- ☐ Architecture declarations
- ☐ Package declarations
- ☐ Package body declarations

3. Do you have the library units referenced by the design?
4. Can the parser parse your design?
5. Have you modified the design to eliminate or change those constructs that VHDL In cannot convert? (See [Chapter 1, “Conversion Issues.”](#))
6. Have you listed all the Cadence libraries required by VHDL In in a `cds.lib` file either in the current working directory or in the standard installation path?
7. For standalone mode line commands
 - ☐ Have you created a VHDL In parameter file?
 - ☐ Have you created a file listing all the VHDL design files? (This file is optional.)

Setting Up the Library Environment

To set up your library environment before you import the example design, follow these steps:

1. Modify your `cds.lib` file to include the five minimum required libraries: `basic`, `sample`, `US_8ths`, `STD (std)`, and `IEEE (ieee)`.

The process for referencing these libraries is described [Chapter 1, “Getting Started with VHDL In.”](#)

2. Start VHDL In from the [CIW menu banner](#) or from the VHDL Tool Box.
3. Create the VHDL design files for the full adder in the directory of your choice.

In this example, the directory used is

```
~/vhdlin_examples/simple_design/
```

Importing to VHDL Cells

In this section, you import the full adder design as a VHDL design [cell](#).

Setting Up the VHDL Import Form

Follow these steps to set up the VHDL Import form for the import process.

1. In the CIW, select *File – Import – VHDL*.

The VHDL Import form opens.

2. Locate the directory for the VHDL source files you want to import.

You can move down a directory in either of the following ways:

- ☐ Enter a path to the directory in the *File Name* field and press *Return*.
- ☐ Double-click a directory in the *Files List* Box.

You can move up one directory any of the following ways:

- ☐ Enter the path to the directory in the *File Name* field and press *Return*.
- ☐ Enter two dots (..) in the *File Name* field and press *Return*.
- ☐ Double-click ../ in the *Files List* Box.

The correct path is displayed below the Files List Box, and the source files are displayed in the Files List Box.

3. Display only files matching a certain pattern, from the current directory. (This is optional.)

Enter the pattern to match with wild card characters in the *File Name* field and press *Return*. The *Files List* Box lists only VHDL files in the directory matching the string you entered in the *File Name* field.

4. Select and add each source file you want to import from the *File Name List* Box to the *Import Files List* Box.

To locate a file name in the *Files List* Box, use the scroll bar to the right of the Files List Box.

You can select a single file and add the file to the Import Files List Box any of the following ways:

- ☐ Double-click the name in the Files List Box.
- ☐ Enter a file name in the *File Name* field and press *Tab*.
- ☐ Enter a file name in the *File Name* field and click *Add>>*.

You can also select multiple files (listed adjacently) and add them simultaneously to the Import Files List Box. Drag the mouse across the file names to highlight them, and click *Add>>*.

5. Press *Tab* to go to the *Target Library Name* field.

Note: When you finish an entry in a field in the VHDL Import form, press *Tab* to go to the next field.

Do not press *Return*. Pressing *Return* is the same as clicking *OK*, which indicates you are ready to import. The form closes when you press *Return* after you specified an

VHDL In for Virtuoso Design Environment User Guide and Reference

Importing a Simple VHDL Design

existing target library. A dialog box with an error message appears when you press *Return* without specifying a target library.

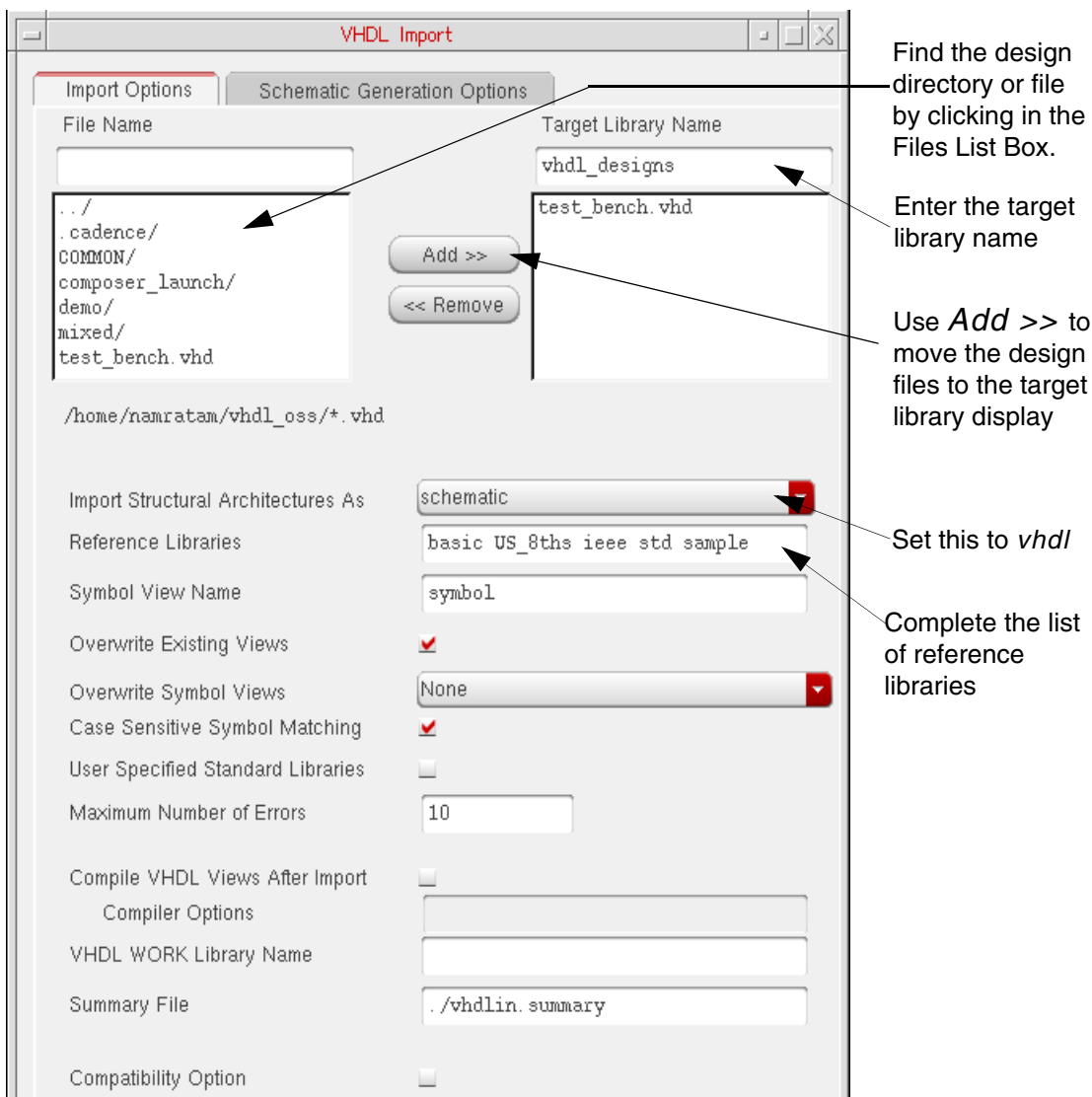
6. In the *Target Library Name* field, enter the target library name that you want to import the source files into (*vhdl_designs*). Press *Tab* to finish.

To accommodate the name mapping scheme used by the ncvhdl parser, use only lowercase letters to name your target libraries.

7. Set the *Import Structural Architectures As* cyclic field to *vhdl*.

You can leave all other options at their default setting.

8. In the *Reference Libraries* field, add *sample*, *std*, and *ieee* to the list.



9. Click *Apply*.

Clicking *Apply* instead of *OK* keeps the form open. You can then start the next example of importing to a netlist without having to complete the list of libraries in the *Reference Libraries* field again or having to add the designs to the Import Files List Box again.

The system imports the design file names into the target library you specified. You can use the VHDL Tool Box form or the Library Manager form to read or edit the cell view of each file.

Creating a New Target Library

If you specify a target library name that does not exist, a dialog box opens, asking you if you want to create the new library. If you click *Yes*, the New Library form opens.

- Enter the library name and the directory where you want the library located.

Put the Cadence versions of the VHDL design, the netlist, and the schematic in *~/vhdl_designs*, *~/netlist_library*, and *~/sch_lib*, respectively.

Select *Do not need process information*, and click *OK*

A message appears in the CIW display area:

```
Created library "vhdl_design" as "/net/.../~/vhdlin_examples/simple_design/vhdl_designs"
```

After the new library is created, you must go back to the [VHDL Import Form](#) and click *Apply* again to import the file names you specified into the target library you just created.

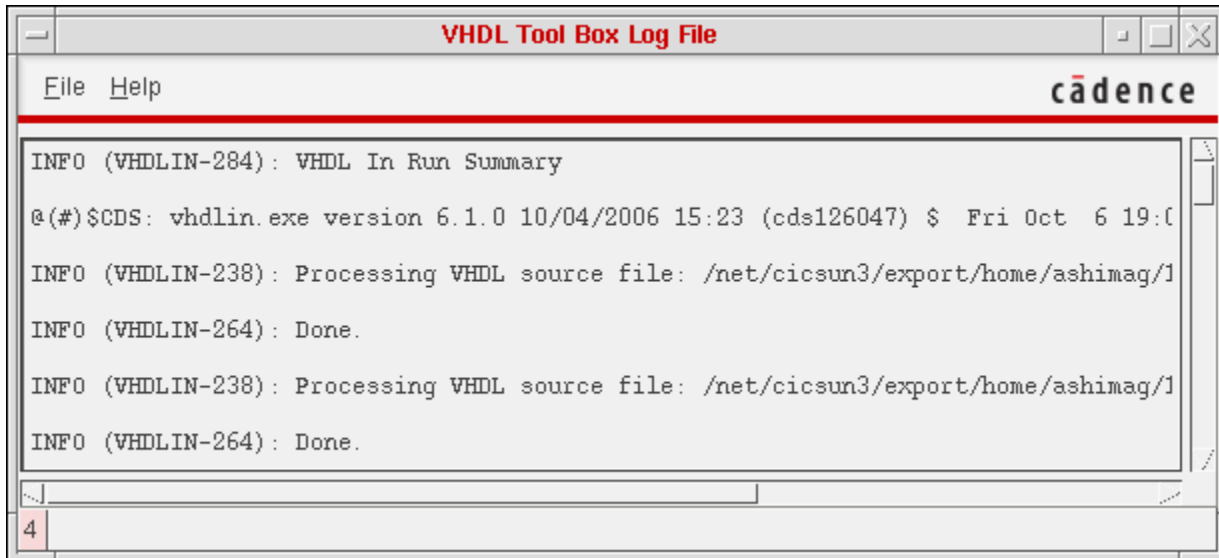
Viewing the Results

A message in the CIW display area tells you that the VHDL Import process has started. When the process finishes, you are presented with a dialog box, which asks if you want to view a summary file. If you click *Yes*, the VHDL Tool Box status window appears, displaying a summary file. You can also view the log of resulting cellviews of your design in the summary file.

Viewing the Summary File

When the import process has finished successfully, you can examine the summary file to check the results of importing to VHDL. The summary file contains information about the number of files processed by VHDL In, the number of files attempted to be imported by VHDL In, the design units imported, and their view types. The summary file also mentions the files that VHDL failed to import, and the reason for the failure.

The following summary file was generated for a successful VHDL to VHDL import process. You can view the summary file through the VHDL Tool Box status window labeled VHDL Tool Box Log File. The window refers to the VHDL Tool Box even if you start from the CIW.



To close the VHDL Tool Box status window

- Select *File – Close*.

The contents of the summary file and the error file are saved with the name you had typed in the *Summary File* field in the VHDL Import Form. The default value of Summary Field is `./vhdlin.summary`.

For examples of error messages, see Chapter 1, “Error Messages.”

- To close the VHDL Tool Box status window, select *File – Close*.

Viewing the Design’s Cellviews

Once the import process has finished successfully, you can examine the cellviews in the target library (such as the *entity* cellview) to check the results of importing to a VHDL cell.

1. In the CIW, select *Tools – Library Manager*.

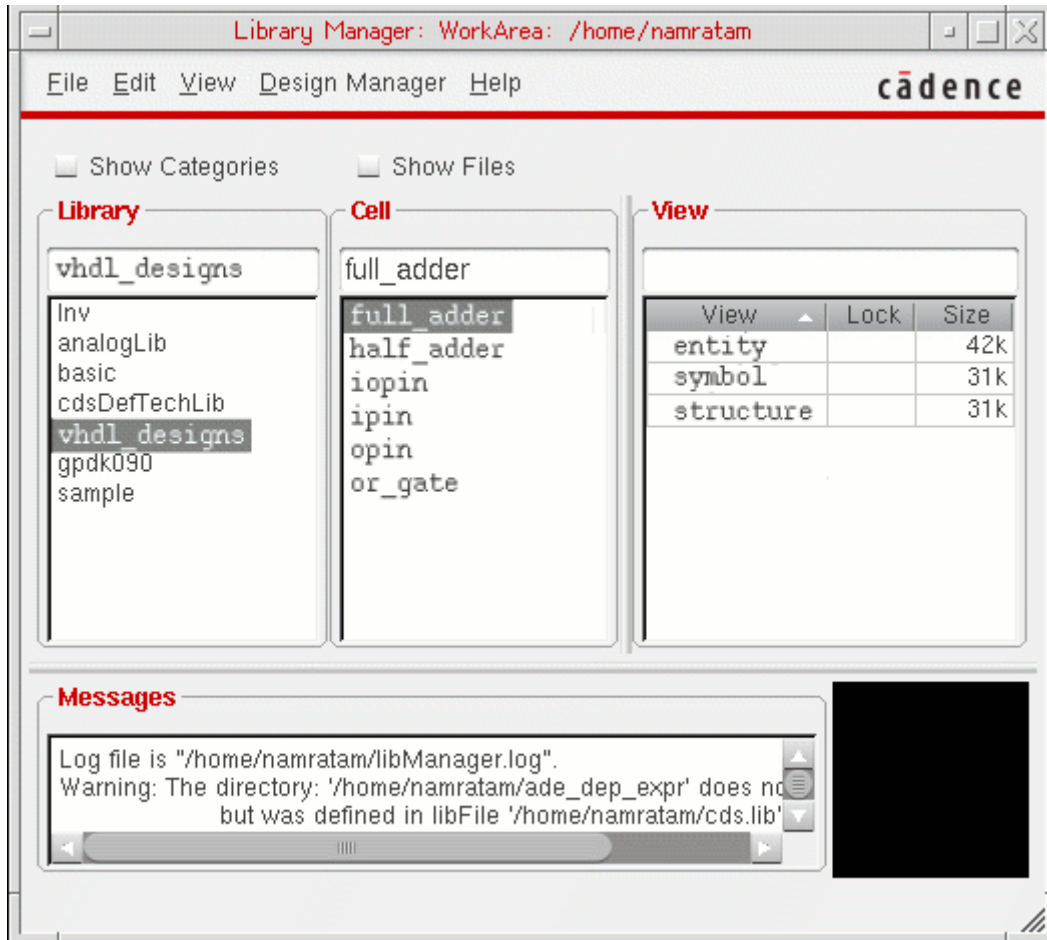
The Library Manager window opens.

2. In the Library list box, click the *vhdl_designs* library name.

The Cell list box displays two cells for the *vhdl_designs* library: *full_adder* and *half_adder*.

3. In the Cell list box, click the *full_adder* cell name.

The View list box displays three cellview names for the full adder design: *entity*, *structure*, and *symbol*.



4. In the View list box, double-click the *entity* cellview file name.

VHDL In for Virtuoso Design Environment User Guide and Reference

Importing a Simple VHDL Design

A text editor window opens, displaying the contents of the *entity* cellview, as shown below: a text file and directory for the *entity* file for the *full_adder* design.

```
-- Created by @(#)CDS: vhdlin.exe version 6.1.0 10/04/2006 15:23 (cds126047) $
-- on Fri Oct  6 19:24:20 2006

entity full_adder is
port(
a : in bit;
b : in bit;
carry_in : in bit;
ab : out bit;
carry_out : out bit
);
end full_adder;
```

You can open the same file in the Language Sensitive Editor, or **LSE** (*emacs*) that highlights the language specific keywords. If you are using LSE and the text editor window is empty, use the *File – Open* command to open `~/vhd1_designs/full_adder/entity`.

VHDL In places VHDL design text files in the `vhd1.vhd` file. You can open the `vhd1.vhd` file for viewing to verify the contents of the file.

5. To open the `vhd1.vhd` file for viewing, double-click the `vhd1.vhd` file name.

The top half of the LSE text editor window displays the contents of the `vhd1.vhd` file, and the bottom half shows the files and directories in the entity directory.

Importing to a Netlist

In this section, you import the full adder design as a OA format netlist. If the VHDL Import form is still open from the first example, [Importing to VHDL Cells](#), you can skip steps 1 through 3.

Setting Up the VHDL Import Form

Follow these steps to set up the VHDL Import form for the import process.

1. In the CIW, select *File – Import – VHDL*.

The [VHDL Import Form](#) opens.

2. In the Files List Box, double-click entries until the design directory is displayed.

To move the display up one directory, click `../`. To display the contents of a directory, click the directory name.

3. In the *Target Library Name* field, enter the name of the target library (*netlist_library*).

To accommodate the name mapping scheme used by the ncvhdl parser, use only lowercase letters to name your target libraries.

4. Set the *Import Structural Architectures As cyclic* field to *netlist*.

You can leave all other options at their default setting.

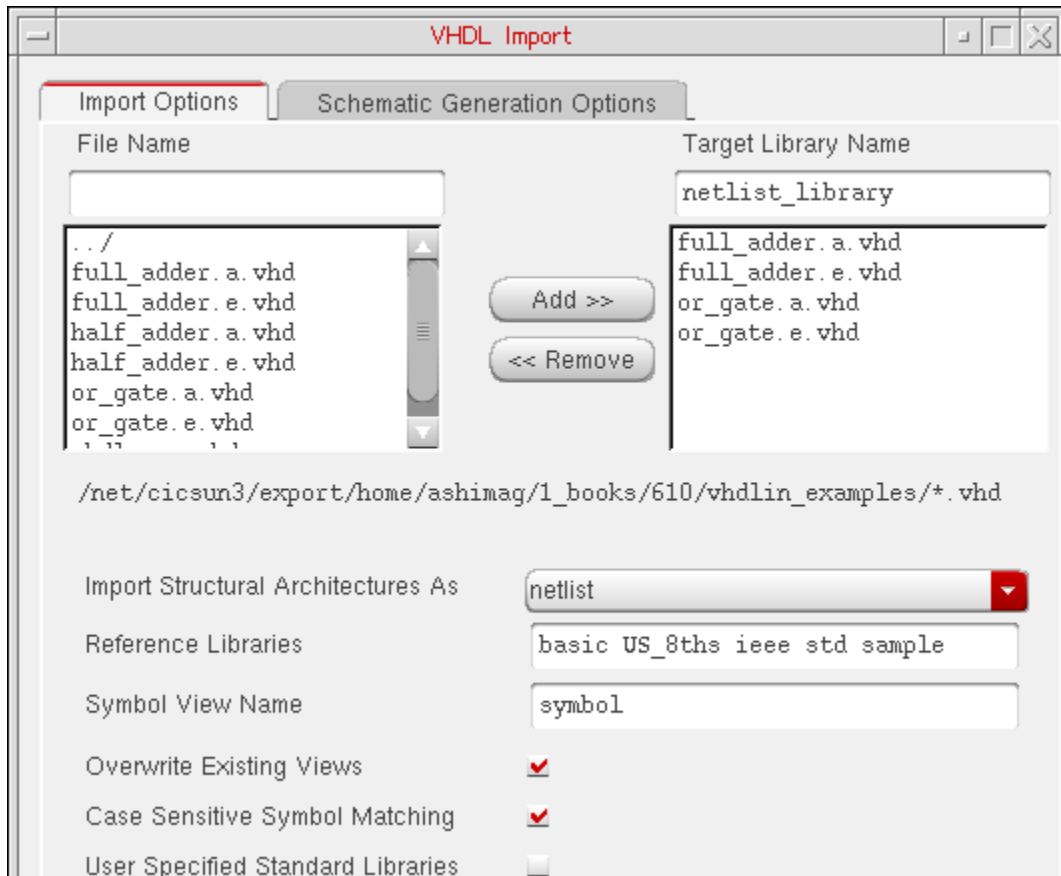
5. Highlight the four file names for the full adder design, and click *Add>>* to add them to the Import Files List Box.

You can also double-click a file name to transfer a file.

Drag the cursor across the file names to highlight multiple files in the box.

6. Click *Apply*.

When you click *Apply* instead of *OK*, the form remains open. You can then start the procedure [Importing to a Schematic](#) at step 3.



Creating a New Target Library

If you specify a target library name that does not exist, a dialog box opens, asking you if you want to create the new library. If you click *Yes*, the New Library form opens.

- Enter the library name and directory, select *Do not need process information*, and click *OK*

A message appears in the CIW display area:

```
Created library "netlist_library" as ".../vhdlin_examples/simple_design/
netlist_library"
```

After the new library is created, you must go back to the [VHDL Import Form](#) and click *Apply* again to begin the import process.

Viewing the Results

A message in the CIW display area tells you that the VHDL Import process has started. When it finishes, you are presented with a dialog box, which asks if you want to view a summary file. If you click **Yes**, the VHDL Tool Box status window appears, displaying a summary file. You can also view the log of resulting cellviews of your design in the summary file.

Viewing the Summary File

When the import process fails, you can examine the errors in the VHDL Tool Box status window labeled *VHDL Errors/Warnings*

For examples of dialog boxes displaying error messages, see [Chapter 1, “Error Messages.”](#)

To close the VHDL Tool Box status window

- Select *File – Close*.

When the import process has finished successfully, you can examine the summary file to check the results of importing to a netlist. The summary file contains information about the number of files processed by VHDL In, the number of files attempted to be imported by VHDL In, the design units imported, and their view types. The summary file also mentions the files that VHDL fails to import, and the reason for the failure.

You can view the summary file through the VHDL Tool Box status window labeled VHDL Tool Box Log File.

```
INFO (VHDLIN-284): VHDL In Run Summary
@(#)SCDS: vhdlin.exe version 6.1.0 10/04/2006 15:23 (cds126047) $ Fri Oct 6
INFO (VHDLIN-238): Processing VHDL source file: /net/cicsun3/export/home/ashi:
INFO (VHDLIN-264): Done.
INFO (VHDLIN-238): Processing VHDL source file: /net/cicsun3/export/home/ashi:
INFO (VHDLIN-264): Done.
```

To close the VHDL Tool Box status window

- Select *File – Close*.

The contents of the summary file are saved with the name you had typed in the *Summary File* field in the VHDL Import Form. The default value of Summary Field is `./vhdlin.summary`.

Viewing the Design's Cellviews

Once the import process has finished successfully, you can examine the cellviews in the target library (such as the *structure* cellview) to check the results of importing to a netlist.

1. In the CIW, select *Tools – Library Manager*.

The Library Manager window opens.

2. In the Library list box, click the *netlist_library library*.

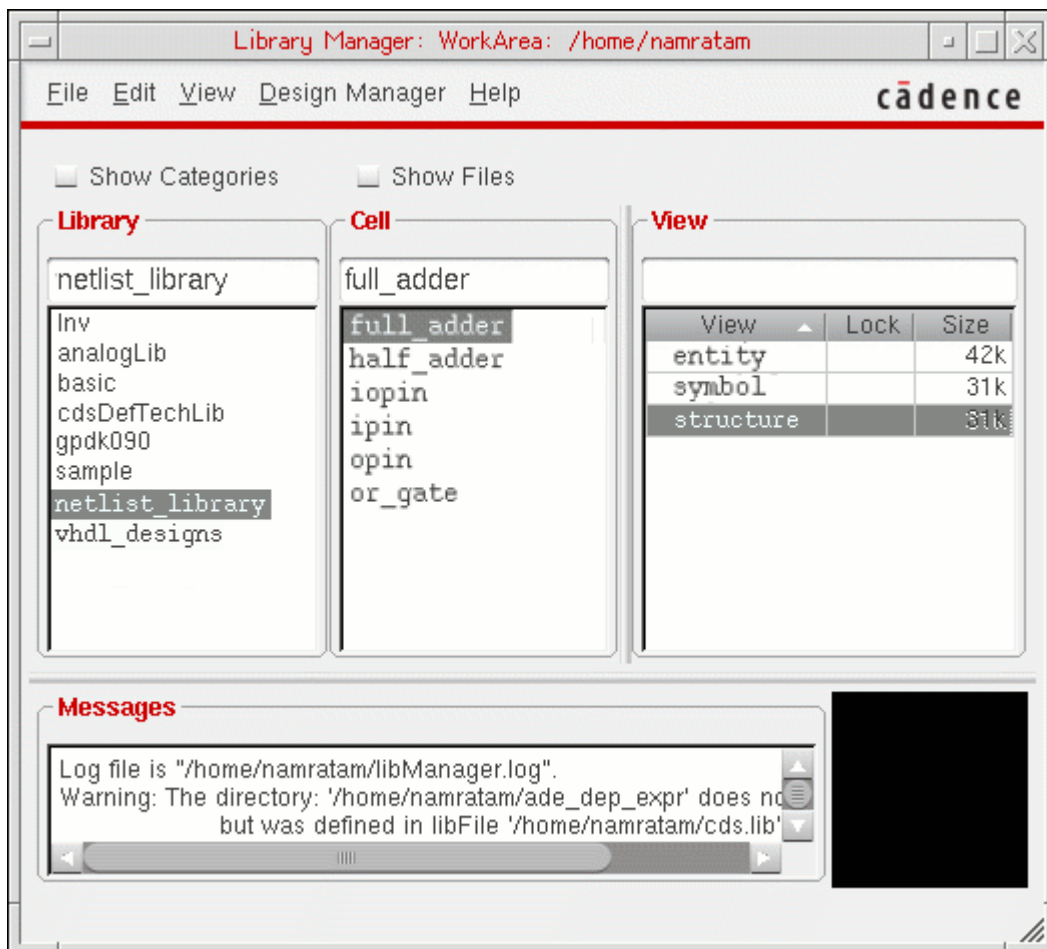
The Cell list box displays three cells: *full_adder*, *half_adder*, and *or_gate*.

3. In the Cell list box, click the *full_adder* cell name to display its cellviews.

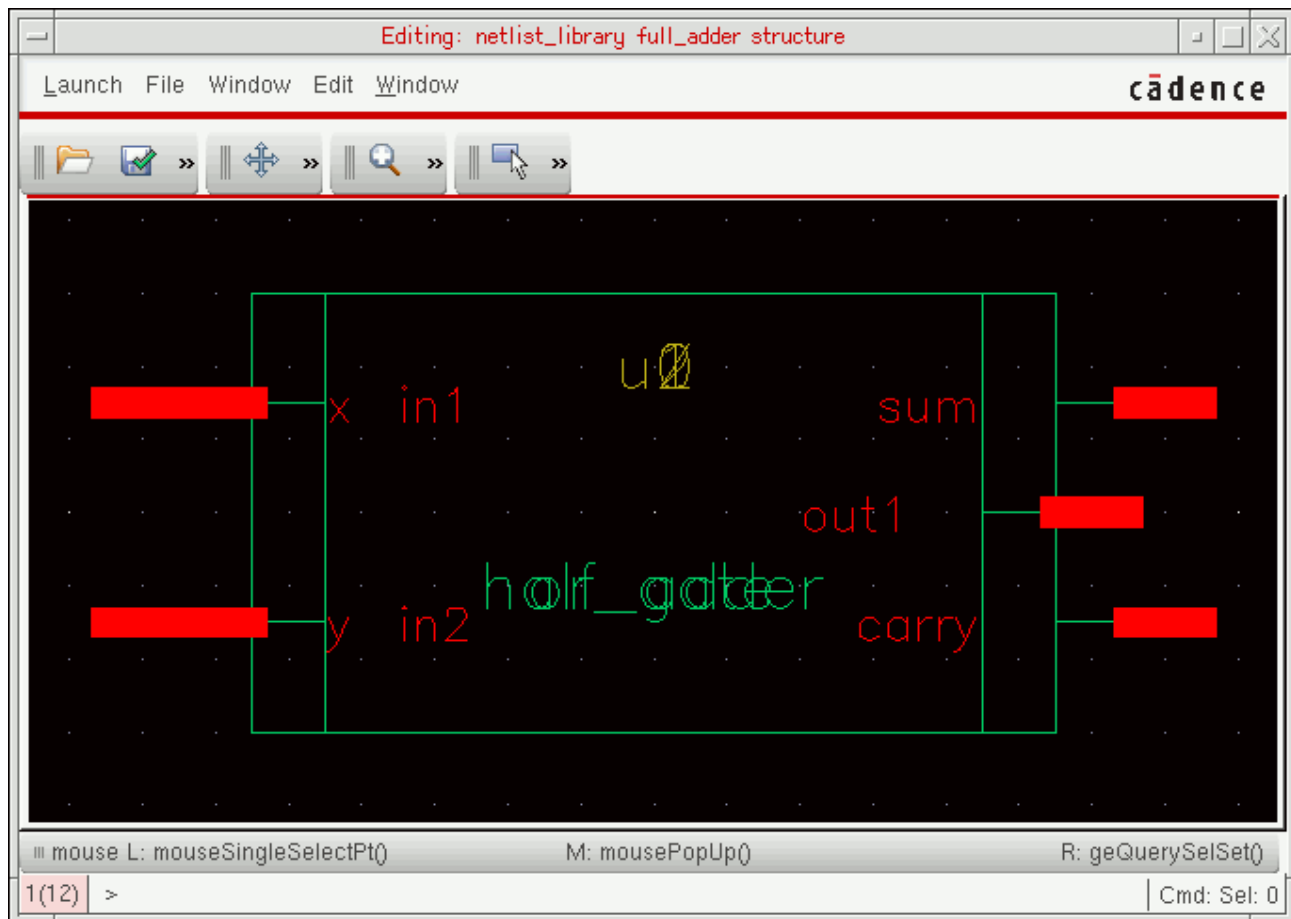
VHDL In for Virtuoso Design Environment User Guide and Reference

Importing a Simple VHDL Design

The View list box displays three cellview names: *entity*, *structure*, and *symbol*



- Click the *structure* cellview name to display the *structure* view as shown below:



VHDL In places the VHDL design netlist in the `netlist.oa`. These files are in the OA format required by Cadence tools. The following is the *structure* directory under the *full_adder* component.

```
tpInx01d:/net/cicsun3/export/home/ashimag/1_books/610/netlist_library/full_adder/structure ls
-|
total 24
-rw-rw-r-- 1 ashimag tp          2880 Oct  6 19:40 data.dm
-rw-rw-r-- 1 ashimag tp           39 Oct  6 19:40 master.tag
-rw-rw-r-- 1 ashimag tp       19456 Oct  6 19:40 netlist.oa
-rw-rw-r-- 1 ashimag tp        689 Oct  6 19:50 netlist.oa.cdslck
tpInx01d:/net/cicsun3/export/home/ashimag/1_books/610/netlist_library/full_adder/structure █
```

Importing to a Schematic

In this section, you import the full adder design as a Virtuoso Schematic Editor L schematic. Start at step 3 if you left the VHDL Import form open from the previous example, [Importing to a Netlist](#).

Setting Up the VHDL Import Form

Follow these steps to set up the VHDL Import form for the import process.

1. In the CIW, select *File – Import – VHDL*.

The Import form closes and the [VHDL Import Form](#) opens.

2. Double-click entries in the *Files List Box* until the design directory is displayed.

To move the display up one directory, click *../*. To display the contents of a directory, click the directory name.

3. Enter the name of the target library (*sch_lib*) in the *Target Library Name* field.

To accommodate the name mapping scheme used by the *ncvhdl* parser, use only lowercase letters to name your target libraries.

4. Highlight the four file names for the full adder design, and click *Add>>* to add them to the Import Files List Box.

You can also double-click a file name to transfer a file.

5. Set the *Import Structural Architectures As* cyclic field to *schematic*.

You can leave all other options at their default setting.

Although you are generating a schematic in this example, you do not have to use the options under the *Schematic Generation Options* tab. You are only using the default settings for schematic generation to import this example design.

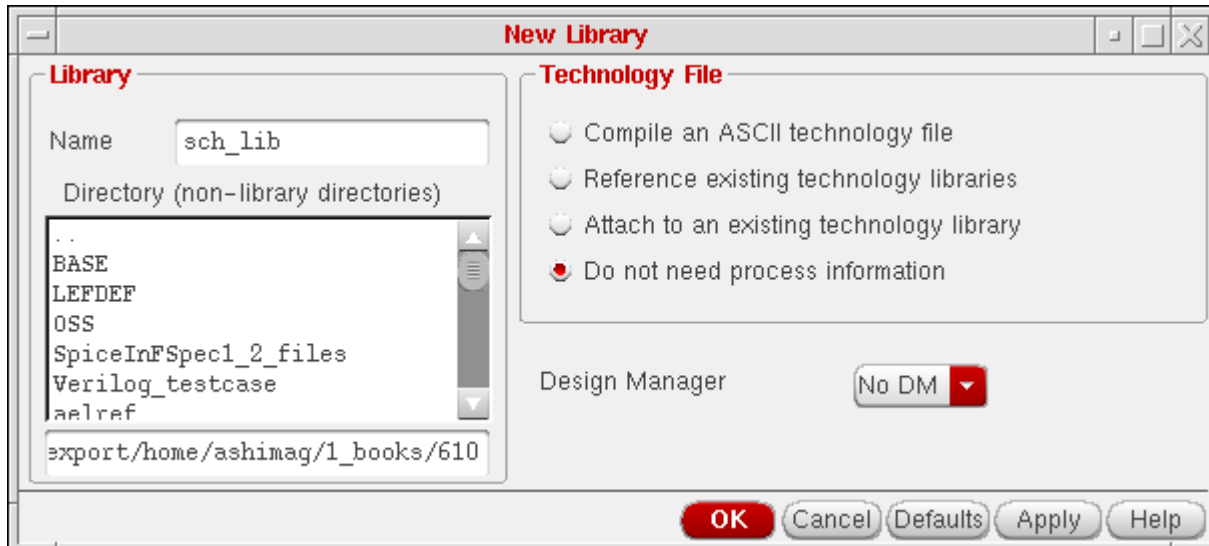
However, the form remembers the last schematic generation options used. So if you have recently performed schematic generation, click the *Schematic Generation Options* bar to open that form and click *Defaults* to reset all the options, then click *OK*.

6. Click *Apply*.

When you click *Apply* instead of *OK*, the form remains open. You can then start the next example at step 5.

Creating a New Target Library

If you specify a target library name that does not exist, a dialog box opens, asking you if you want to create the new library. If you click *Yes*, the New Library form opens.



- Enter the library name and directory, select *Do not need process information*, and click *OK*.

A new library is created, and VHDL files imported. There is no need to go back to the VHDL Import form to click *Apply* again.

Viewing the Results

A message in the CIW display area tells you that the VHDL Import process has started. When it finishes, you are presented with a dialog box, which asks if you want to view a summary file. If you click *Yes*, the VHDL Tool Box status window appears, displaying a summary file. You can also view the log of resulting cellviews of your design in the summary file.

Viewing the Summary File

When the import process fails, you can examine the errors in the VHDL Tool Box status window labeled VHDL Errors/Warnings.

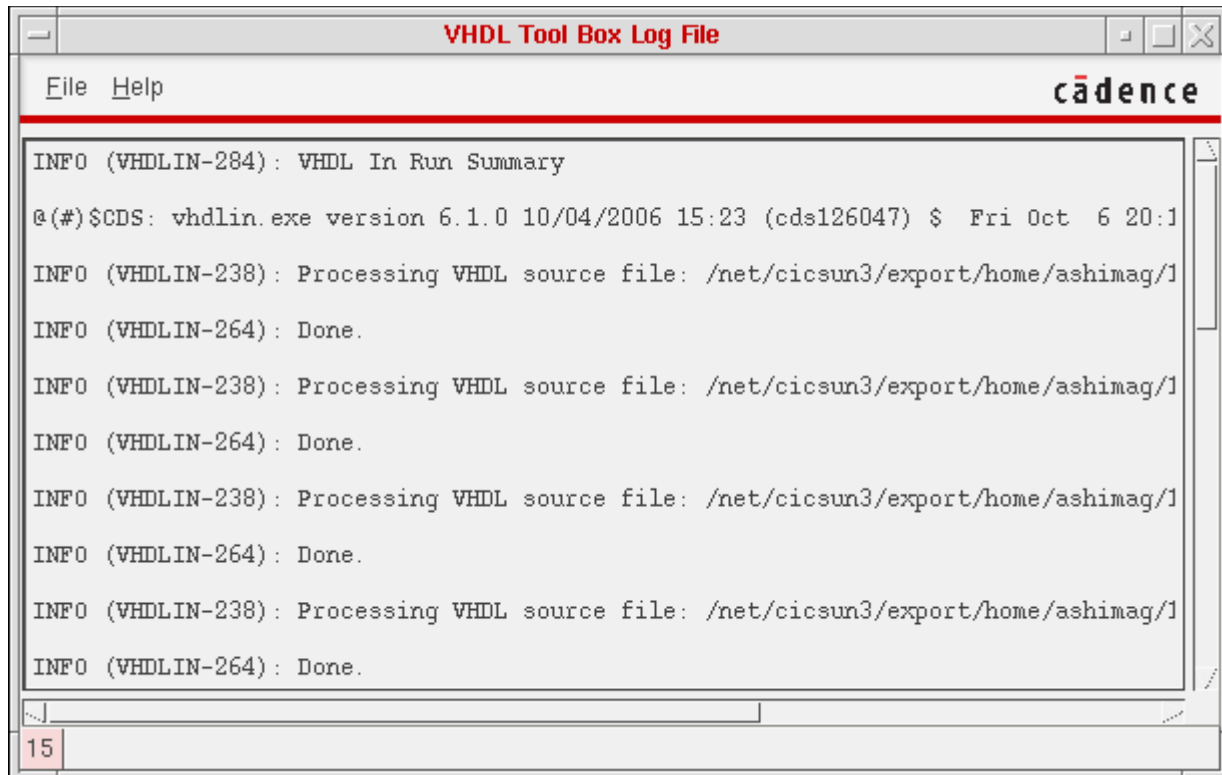
For examples of dialog boxes displaying error messages, see Chapter 1, “Error Messages.”

To close the VHDL Tool Box status window

- Select *File – Close*.

When the import process has finished successfully, you can examine the summary file to check the results of importing to a schematic. The summary file contains information about the number of files processed by VHDL In, the number of files attempted to be imported by VHDL In, the design units imported, and their view types. The summary file also mentions the files that VHDL failed to import, and the reason for the failure.

You can view the summary file through the VHDL Tool Box status window labeled *VHDL Tool Box Log File*. When the import process finishes, the VHDL Tool Box status window display of the summary file looks similar to this.



To close the VHDL Tool Box status window

- Select *File – Close*.

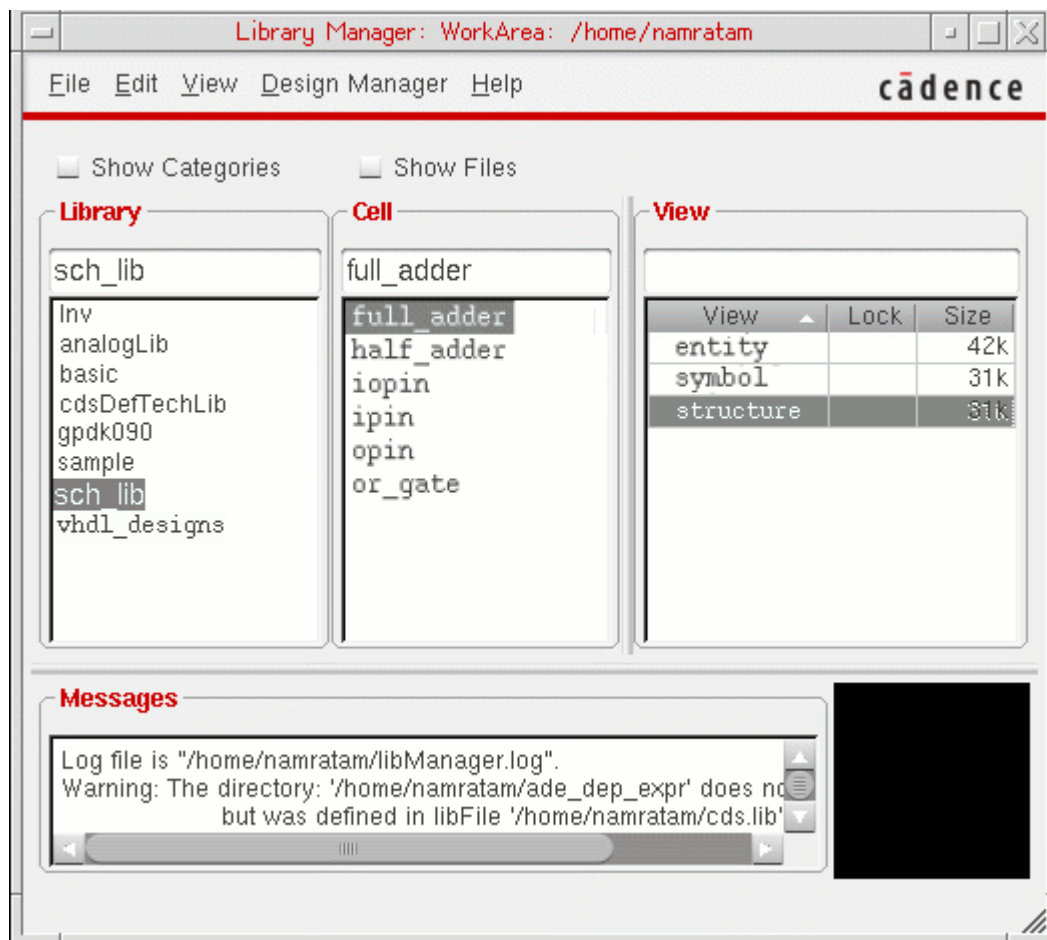
The contents of the summary file are saved with the name you had typed in the *Summary File* field in the *VHDL Import Form*. The default value of Summary Field is `./vhdlin.summary`.

Viewing the Design's Cellviews

Once the import process has completed successfully, you can examine the cellviews in the target library (such as the *structure* cellview) to check the results of importing to a schematic.

1. In the CIW, select *Tools – Library Manager*.

The Library Manager window opens.



2. In the Library list box, click the *sch_lib* library name.

The Cell list box displays three cells for the *sch_lib* library: *full_adder*, *half_adder*, and *or_gate*.

3. In the Cell list box, click the *full_adder* cell name.

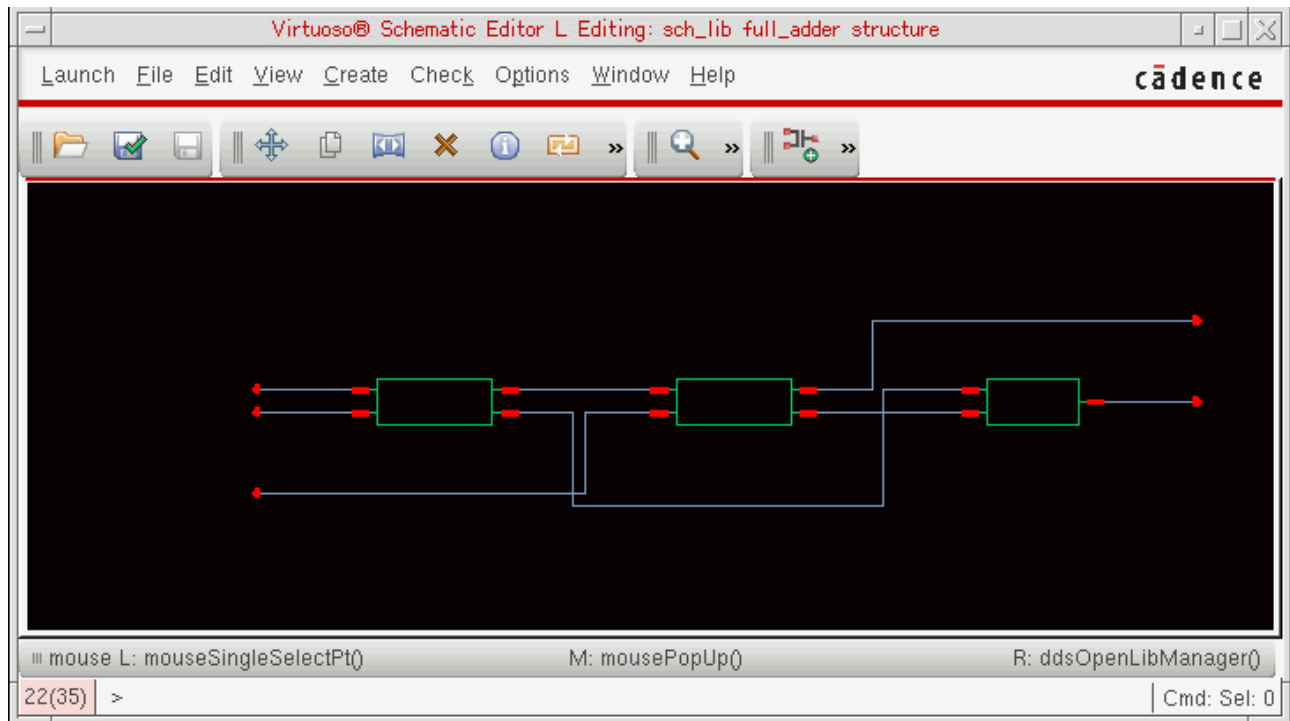
The View list box displays three cellview names for the full adder design: *entity*, *structure*, and *symbol*.

4. In the View list box, double-click the *structure* cellview file name.

VHDL In for Virtuoso Design Environment User Guide and Reference

Importing a Simple VHDL Design

The Virtuoso Schematic Editor L opens, displaying the contents of the *structure* cellview: a schematic for the *full_adder* design. The following figure shows the structure view of the *full_adder* cell: a Virtuoso schematic for the VHDL adder design.



This schematic results from using the default values for the schematic generation options.

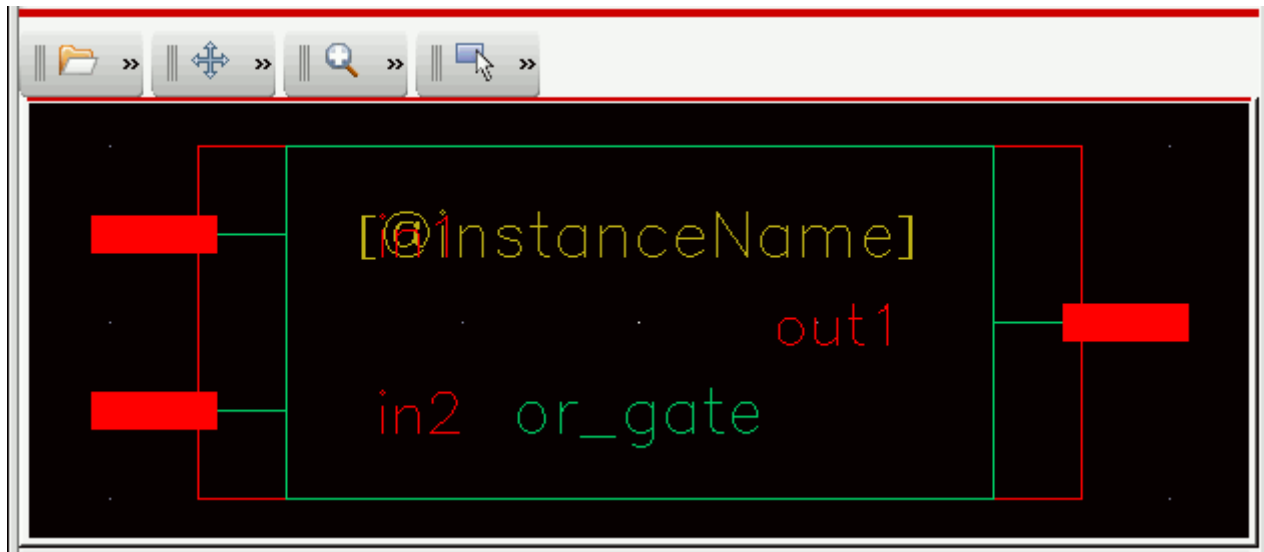
You can also select the symbol view of any design element and view and edit the element symbol in the Virtuoso Symbol Editor L window.

5. In the Library Manager window, click the *or_gate* cell, and double-click its symbol view.

VHDL In for Virtuoso Design Environment User Guide and Reference

Importing a Simple VHDL Design

The Virtuoso Symbol Editor L window opens, displaying the symbol generated by VHDL In for the OR gate.



Importing a Complex VHDL Design (RISC Processor Unit)

This chapter discusses the following:

- [Overview](#)
- [RPU Design Structure](#)
- [Setting Up the VHDL Import Form](#)
- [Specifying Schematic Generation Options](#)
- [Creating a Target Library](#)
- [Viewing Error and Log Files](#)
- [Viewing the Results](#)

Overview

This chapter describes how you can import a larger, more complex VHDL design. This example uses the design of an RPU (a Reduced Instruction Set Computer (RISC) Processor Unit). The more complex design of the RISC processor unit (RPU) raises some issues that did not come up with the simple adder design. As part of the process, you use the options under the *Schematic Generation Options* tab to adjust the appearance of the final schematic.

Before you begin this example, use the Library Manager to modify your `cds.lib` file to include the five minimum required libraries, `basic`, `sample`, `US_8ths`, `std`, and `ieee`. The process for referencing these libraries is described in [Chapter 1, “Getting Started with VHDL In.”](#)

RPU Design Structure

This sample design is included in the Cadence software hierarchy, at `<dfll_install_dir>/tools/dfll/samples/vhdlin/Test3`, where `<dfll_install_dir>` is dfll installation directory. Copy this design directory into your own account. In the following example, the RPU design starts in the `~/vhdlin_examples/rpu_design` directory.

The RPU design consists of I/O connection pads and the main design.

The I/O pads include

- `ipad_1` (input pad)
- `opad_1` (output pad)
- `iopad_1` (bidirectional pad)

The design includes 11 unique functional elements:

- `reg16` (16-bit register)
- `reg16c` (reg16 with carry)
- `cntr16` (16-bit counter)
- `mux2` (2:1 multiplexer)
- `dff` (D flip-flop)
- `alu` (Arithmetic Logic Unit)
- `clockgen` (clock source)

- reg4c (4-bit register with carry)
- instructionDecoder (an array of logic)
- regArray (local data storage)
- mux4 (4:1 multiplexer)

Setting Up the VHDL Import Form

Although you can run VHDL In from VHDL Tool Box, this example starts from the CIW. The example design is located in `/usr1/vhdlin/example`. Before you start, copy the design example from the `<dfll_install_dir>/tools/dfII/samples/vhdlin` directory and keep it in your own account.

1. In the CIW, select *File – Import – VHDL*

The VHDL Import form opens.

2. Double-click entries in the Files List Box until the design directory is displayed.

To move the display up one directory, click `../`. To display the contents of a directory, click the directory name.

3. In the *Target Library Name* field, enter the name of the target library (`rpulib`).
4. Set the *Import Structural Architectures As cyclic* field to *schematic*.

You can leave all other options at their default setting.

5. To add the design's entity and architecture files to the Import Files List Box, double-click each file name in the Files List Box.

In this example, the entity and architecture for each component are contained in a single `*.vhd` file.

When you add files to be imported, keep the following items in mind.

- ☐ The VHDL Import form shows only the `.vhd` files in each directory. If you are using other tools or techniques, some of your directories might include C model files and Verilog files.

Even though these files end in `.vhd`, do not import them. VHDL In processes only VHDL files.

- ☐ As you add more and more design files from the design subdirectories, you might discover that different design elements use the same file names.

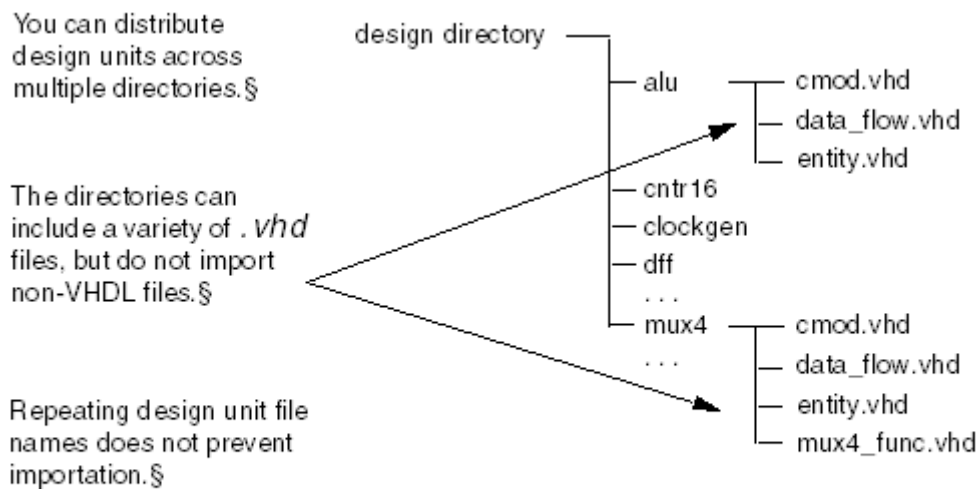
VHDL In for Virtuoso Design Environment User Guide and Reference

Importing a Complex VHDL Design (RISC Processor Unit)

For example, you might call all the architecture files `data_flow.vhd`, and all the entity files `entity.vhd`, and put both sets of files under directories named after the component. The VHDL Import form keeps track of which directory each file comes from and can tell the different files apart.

- ❑ If you cannot remember if you have added a file, select it again, because VHDL In ignores multiple requests to add the same file.

The following figure illustrates and summarizes these three considerations.



Considerations in adding files for import

Specifying Schematic Generation Options

Now you are ready to set the schematic generation options. The three-element adder design is so simple that a schematic generated with just the default settings still looks reasonable. The RPU design, however, is larger and more difficult to place properly.

If you do not set the schematic generation options, VHDL In draws the entire design onto one schematic sheet of infinite size.

1. On the VHDL Import form, click the *Schematic Generation Options* tab.
2. Set *Sheet Border Size* to *C*.
3. In *Maximum Number of Rows*, enter 3.
4. In *Maximum Number of Columns*, enter 6.
5. Set the *Component Density* slider to 15.

The last three options specify a reasonable number of large components that can be placed comfortably on a C-size schematic.

6. Deselect *Generate Square Schematics*.

VHDL In for Virtuoso Design Environment User Guide and Reference

Importing a Complex VHDL Design (RISC Processor Unit)

The standard Cadence schematic sheet frame is rectangular, so turning off this option improves the use of schematic sheet space

The screenshot shows the 'VHDL Import' dialog box with the 'Schematic Generation Options' tab selected. The 'Import Options' tab is also visible. The 'Schematic Generation Options' section includes the following settings:

- Sheet Border Size: C (dropdown menu)
- Maximum Number of Rows: 3 (text field)
- Maximum Number of Columns: 6 (text field)
- Font Height: 0.0625 (text field)
- Line To Line Spacing: 0.2 (text field)
- Line To Component Spacing: 0.5 (text field)
- Component Density: 15 (slider, ranging from Lowest to Highest)
- Pin Placement** (section header)
 - ☒ Left and Right Sides
 - ☐ All Sides
 - ☐ Pin Placement File
 - Pin Placement File Name: (text field)
- Text to Symbol Generator Files: (text field) ... (button)
- Full Place and Route: ☒ Optimize Wire Label Locations ☒
- Generate Square Schematics: ☐ Extract Schematics ☒
- Minimize Crossovers: ☒ Ignore Extra Pins On Symbol ☐
- Fast Schematic Generation** (section header)
 - Generate Fast Schematic: ☒
 - Instances Greater Than: 20000 (text field)
 - Ports Greater Than: 5000 (text field)

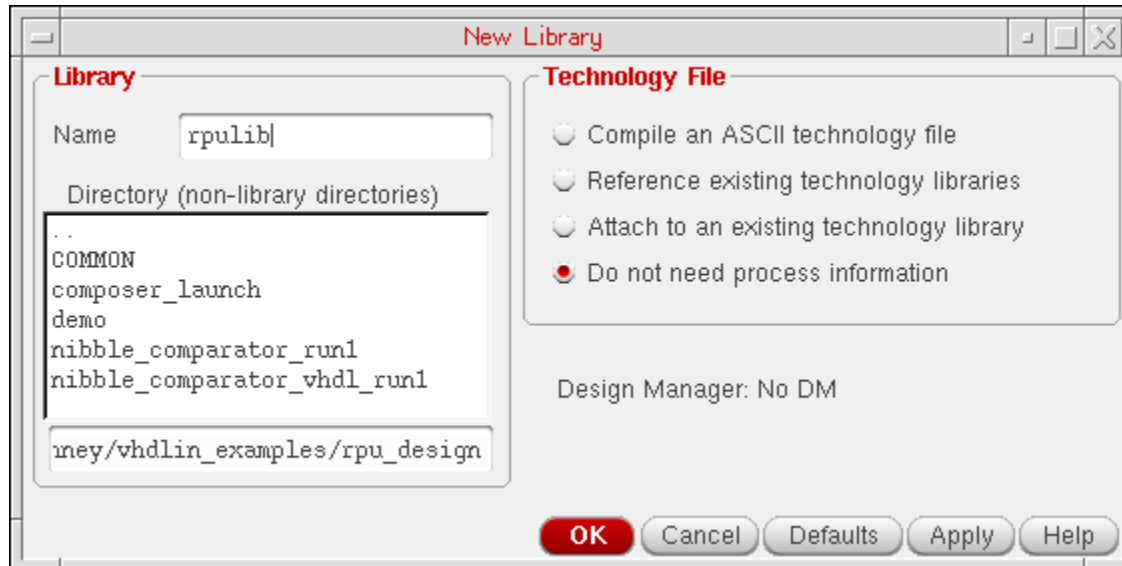
7. Click **OK**.

The form closes.

8. In the VHDL Import form, click **OK** or **Apply**.

Creating a Target Library

Because you specified a target library name in the VHDL Import form that does not exist (rpulib), a dialog box opens, asking you if you want to create the new library. When you click **Yes** in the dialog box, the New Library form opens.



1. Enter the library name and directory.
2. Select *Do not need process information*.
3. Click **OK**.

A message appears in the CIW display area after the new library is created:

Created library "rpulib" as ".../vhdlin_examples/rpu_design/rpulib"

Viewing Error and Log Files

A message in the CIW display area tells you that the VHDL Import process has started. When it finishes, you are presented with a dialog box. If the import process is successful, the box asks if you want to view the log file. If the import process fails, the box asks you if you want to view the error message file. For examples of these and other dialog boxes, refer to [Chapter 7, "Error Messages."](#)

If you receive an error message, go back to the VHDL Import form. After fixing the problems indicated, select a new list of design files to import and run VHDL In again.

The following figure is the VHDL Tool Box status window display of the beginning of the log file for this VHDL import example. The window refers to the VHDL Tool Box even when you start the software from the CIW. Select *File – Close* to close the log window.

Viewing the Results

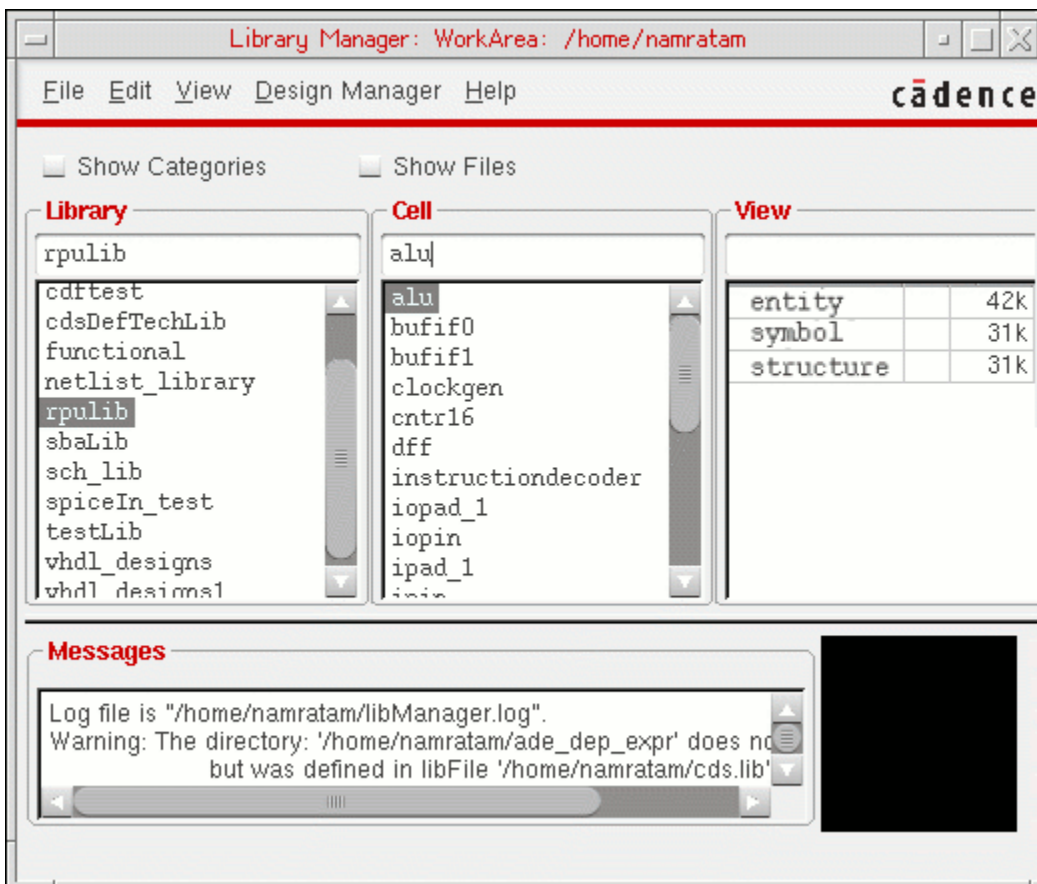
Once the import process has finished successfully, you can examine the target library to check the result.

Accessing the Symbol Editor

You can view the symbol view of a cell in the design you just imported. In this procedure you view the symbol for the *alu* in the imported design.

1. In the CIW, choose *Tools – Library Manager*.

The Library Manager window opens.



VHDL In for Virtuoso Design Environment User Guide and Reference

Importing a Complex VHDL Design (RISC Processor Unit)

2. In the Library column of the Library Manager window, enter *rpulib* in the *Library* text field and press *Return* or click the *rpulib* library.

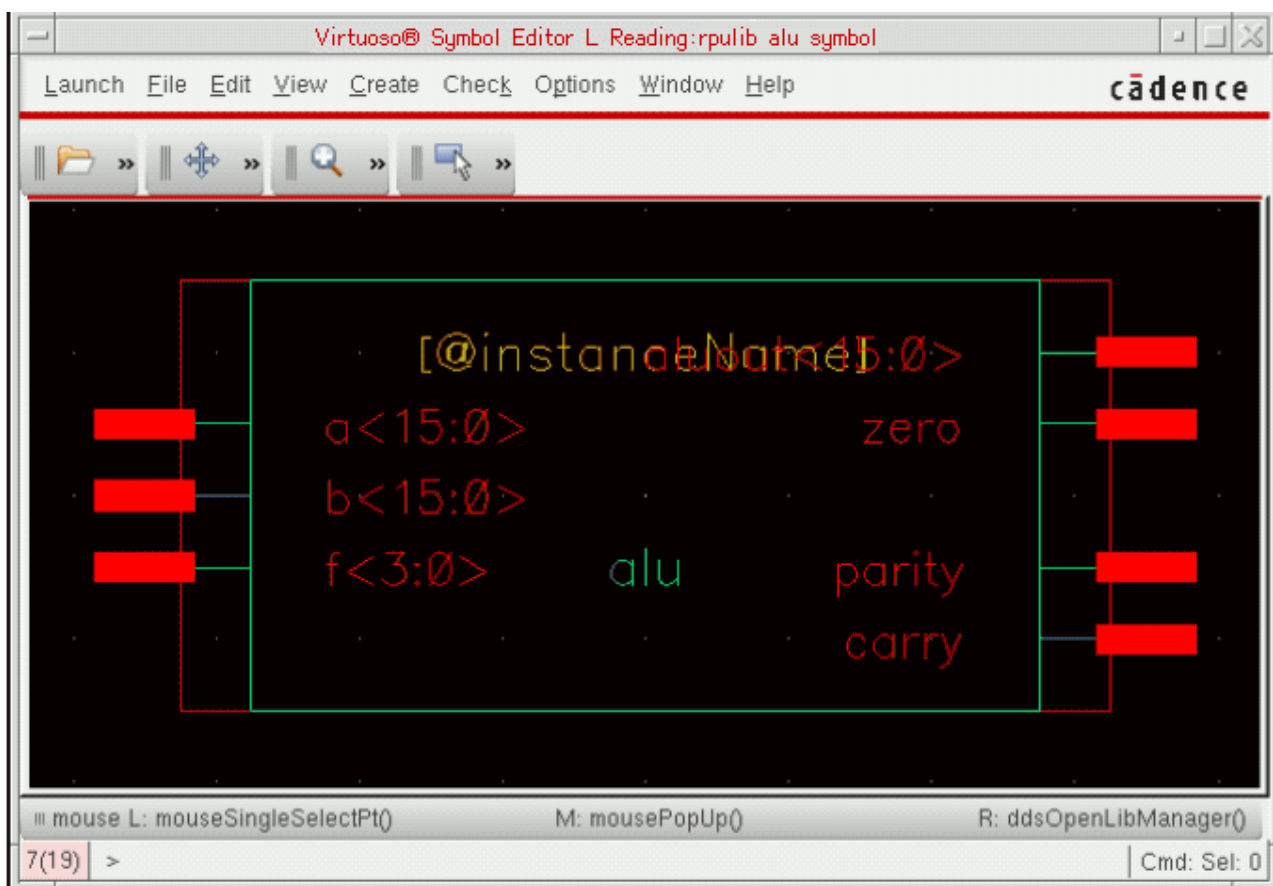
The Cell column displays all the cells of the RPU design elements.

3. To display the cellviews of the alu design, click the *alu* cell name.

The View column displays three cellview names: *dataflow*, *entity*, and *symbol*.

4. Double-click the view name *symbol*.

The Symbol Editor opens, displaying the symbol for *alu*.



Each element in this design now has its own QA symbol. If you do not like how the symbols look, you can edit them.

5. To close the Symbol Editor, select *File – Close*.

Viewing the Schematic

In this procedure, you examine the schematic views that VHDL In has produced for *rpustruc* and *rputop* designs. *rpustruc* is just the RISC processor design itself. *rputop* includes *rpustruc* surrounded by I/O pads.

VHDL In converts all capital letters in the cellview names to lowercase in the schematic.

1. In the CIW, select *Tools – Library Manager*.

The Library Manager window opens.

2. In the Library list box, click the *rpulib* library name.

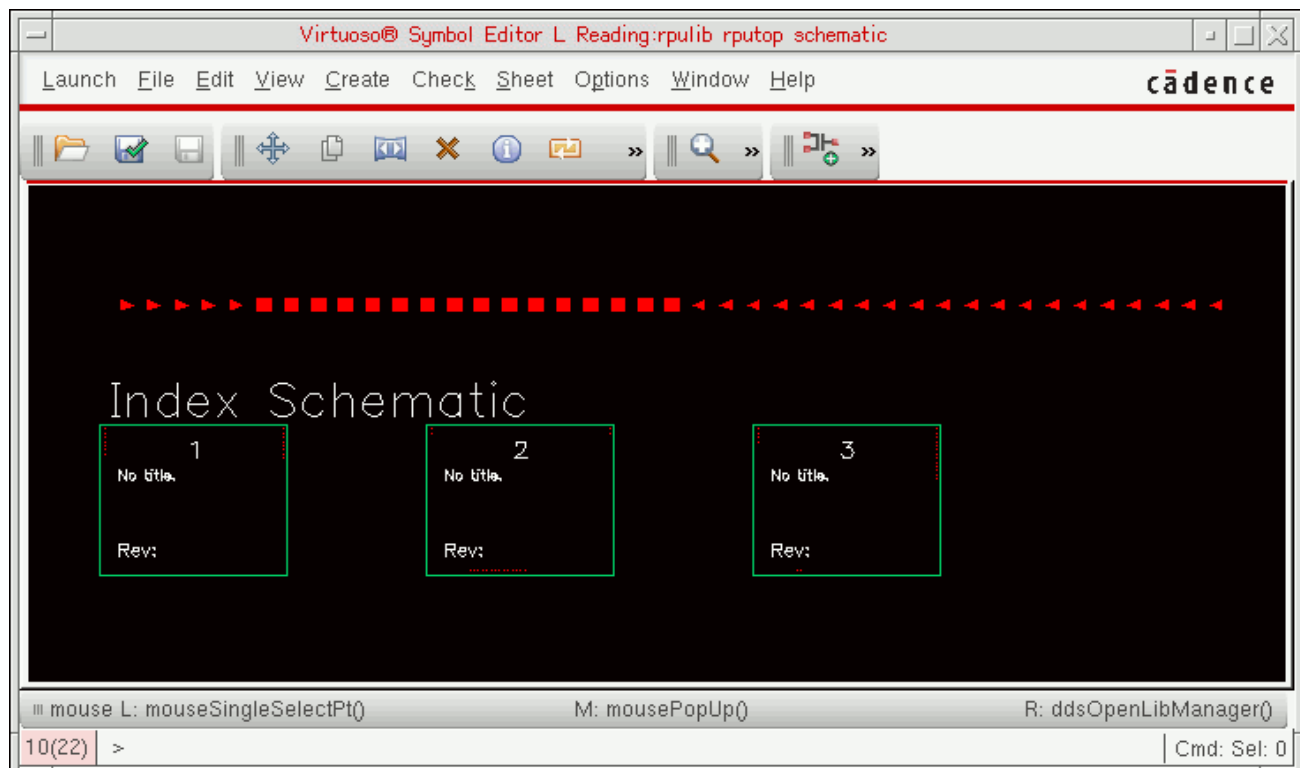
The Cell list box displays nine cells for the *rpulib* library.

3. In the Cell list box, click the *rputop* cell.

The View list box displays three cellview names for the RPU design: *entity*, *schematic*, and *symbol*.

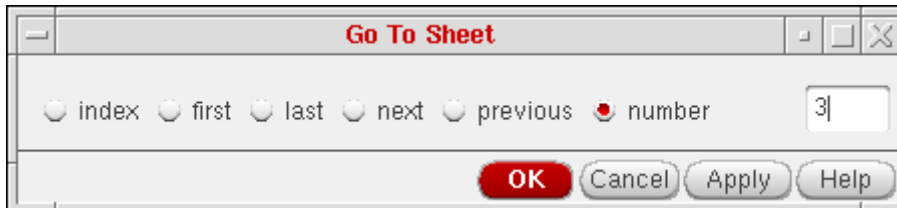
4. In the View list box, double-click the *schematic* cellview of the *rputop* cell to view the schematic of the cell.

The Virtuoso Schematic Editor L opens, displaying the schematic of *rputop*.



VHDL In, following the criteria set up under the *Schematic Generation Options* tab, has divided the top-level design into four sheets. Sheets 1 through 3 include only I/O pads.

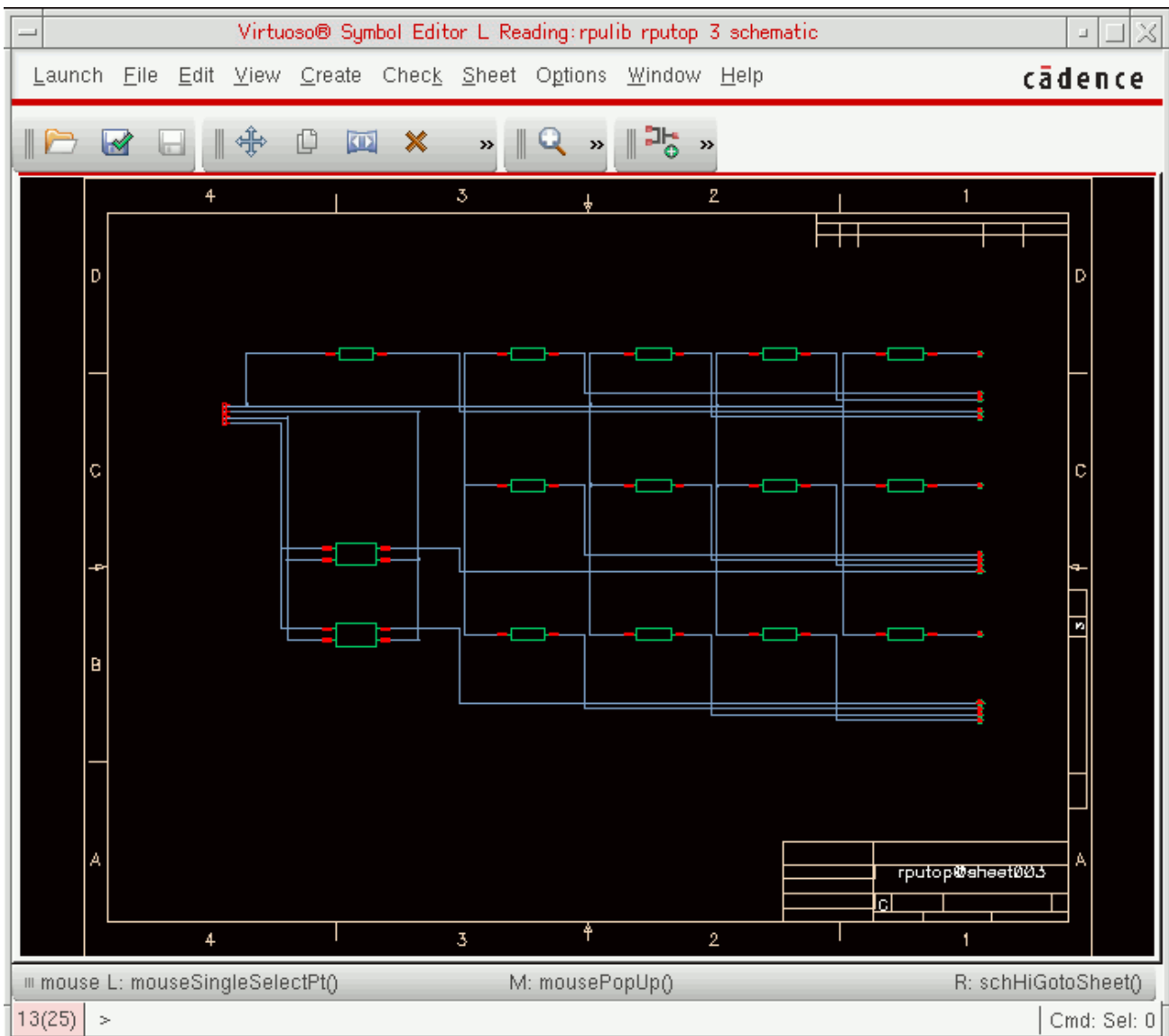
5. In the schematic window, select *Sheet – Go To*, and select Sheet 3.



VHDL In for Virtuoso Design Environment User Guide and Reference

Importing a Complex VHDL Design (RISC Processor Unit)

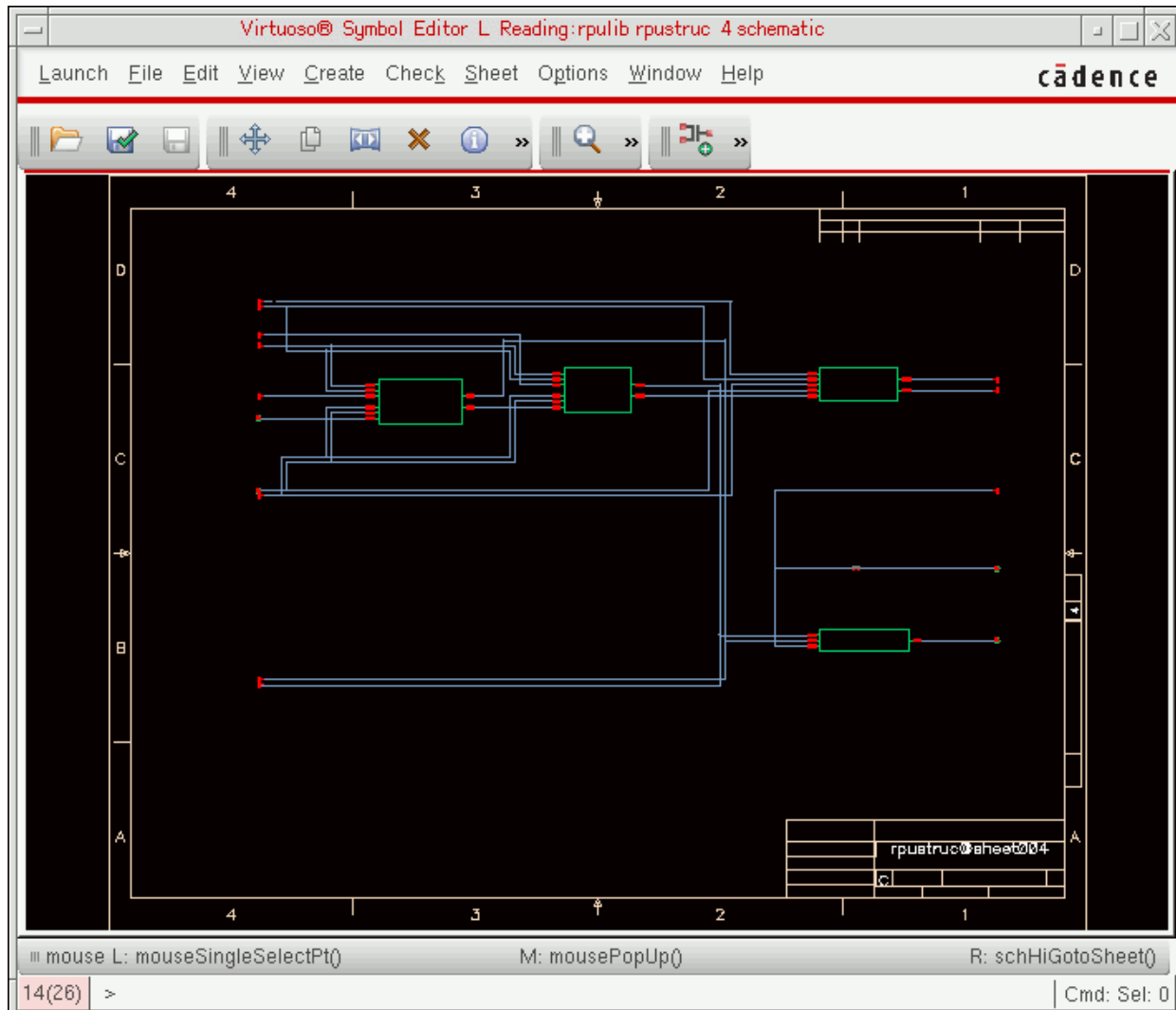
Schematic sheet 3 for rputop is shown below:



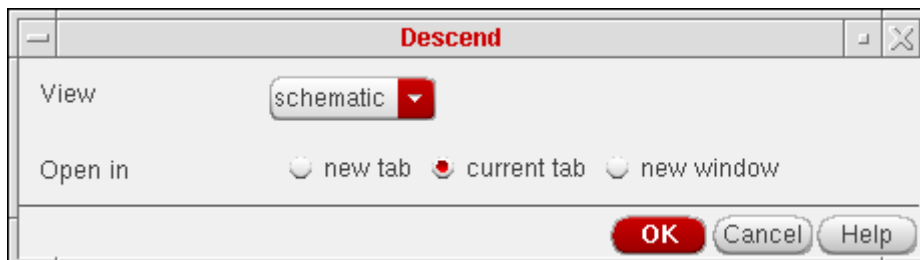
VHDL In for Virtuoso Design Environment User Guide and Reference

Importing a Complex VHDL Design (RISC Processor Unit)

- From the Library Manager double-click the *schematic* view for *rpustruc@sheet004*. This sheet displays *rpustruc* and some I/O pins.



- In the schematic window for the *rpustruc* index schematic, select *Sheet 5*.
- Click *Edit – Hierarchy – Descend Edit*, and click OK from the Descend form.

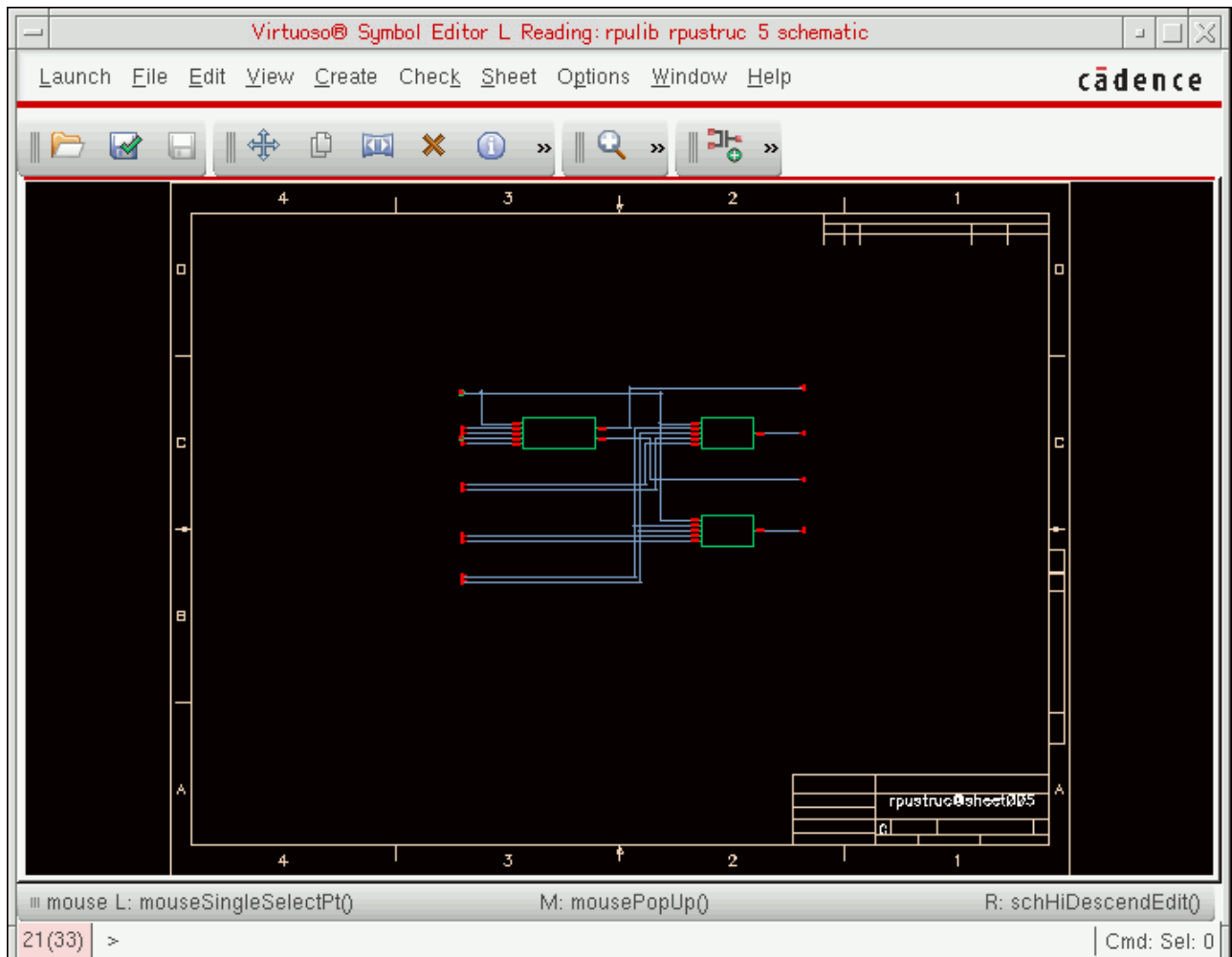


VHDL In for Virtuoso Design Environment User Guide and Reference

Importing a Complex VHDL Design (RISC Processor Unit)

The Virtuoso Schematic Editor L opens Sheet 6 of *rpustruc*.

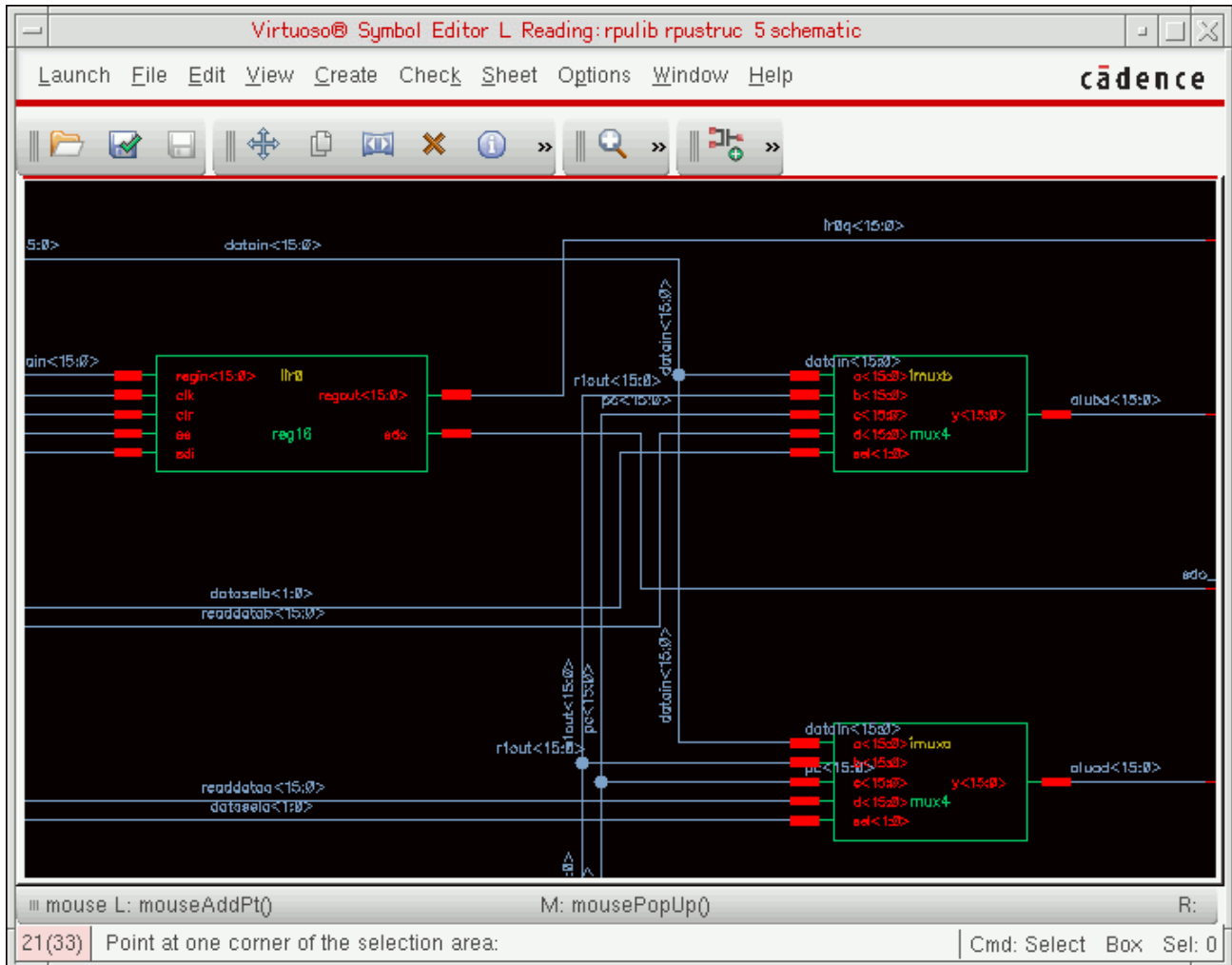
The following figure shows how VHDL In has positioned a portion of the RPU design in a C-size schematic frame and entered the design name in the information section.



VHDL In for Virtuoso Design Environment User Guide and Reference

Importing a Complex VHDL Design (RISC Processor Unit)

To see how VHDL In has routed nets and placed labels, press **z** to zoom in on one section of the schematic.



If you are dissatisfied with the decisions VHDL In has made, you can rerun VHDL In with different schematic generation options, or you can use the schematic editor to make changes to the schematic.

If you descend further into one of the design elements, you can view the entity and dataflow ([architecture](#)) text files.

- ➔ To exit the design, select *File – Close*.

Conversion Issues

This chapter discusses the following:

- [Introduction](#)
- [Binding Issues](#)
- [Case Sensitivity Issues](#)
- [Other Issues](#)
- [Nontranslatable Structural VHDL Constructs](#)

Introduction

Some constructs in VHDL do not map clearly and directly to an equivalent element in the QA format. In these cases, certain issues arise when VHDL In converts the data. This chapter discusses the issues and how VHDL In handles each case.

Binding Issues

A VHDL design can consist of components represented by component declarations that are bound to the entity/architecture pairs. In a design unit, a component instance can be unbound, partially bound, or fully bound to entity/architecture pairs. Component declarations can reside in a referred package that can be used across multiple designs.

VHDL allows a design to be analyzed without all components being bound. A design is ready for import only if all of its component instances are bound or if there is a list of reference libraries that you specify to pick up components that are not bound.

If you do not specify the binding of a component and VHDL In cannot find it in any reference library, VHDL In creates a symbol cellview in the target library. VHDL In uses this new symbol cellview in the schematic view generation of the design that instantiates the component. VHDL In also creates a symbol cellview in reference libraries if the library to which the component instance is bound does not have a symbol cellview. This feature is available when the parameter `writeInrefLibs` is set to TRUE in the parameter file.

Note: You have write permissions in the specified library.

VHDL In creates a symbol cellview for unbound components only if there is no consistent symbol cellview for the cell in the list of reference libraries and the target library. VHDL In searches for the symbol cellview in the target library first.

The two levels of binding produce ambiguities in designs. The examples that follow include descriptions of how the import process handles each case.

One Component, Multiple Entity/Architecture Pairs

The following is an example of multiple instances of the one component bound to multiple entity/architecture pairs.

```
COMPONENT a IS
PORT (
    input: IN std_logic
    output: OUT std_logic
);
```

```
FOR INSTANCE1: A USE ENTITY WORK.Ent1(Arch1);
FOR INSTANCE2: A USE ENTITY WORK.Ent1(Arch2);
FOR OTHERS: A USE ENTITY WORK.Ent2(Arch2);
```

VHDL In resolves the example as follows:

- If multiple instances of a component are bound to multiple entities, then VHDL In instantiates different cells. This is similar to instantiating different components.
- If multiple instances of a component are bound to multiple architectures of the one entity, then VHDL In instantiates the same cell.

For every instance, VHDL In adds the following property to specify which entity/architecture pair an instance is bound to:

```
Property Name: vhdlArchitectureName
Property Type: string
Property Value: libraryName.entityName(architecture name)
```

An example property name is *design_lib.nand2(rtl)*.

One Entity, Multiple Components

The following is an example of instances of multiple components bound to one entity.

```
ENTITY and2 IS
PORT (
    in1      : IN std_logic;
    in2      : IN std_logic;
    Result    : OUT std_logic
);
END and2;
-- BINDING SPECIFICATIONS
COMPONENT sn74And2
PORT (in1, in2: IN std_logic;
      Result: OUT std_logic
);
END COMPONENT
COMPONENT dn84And2
PORT (in1, in2: IN std_logic;
      Result: OUT std_logic
);
FOR ALL: sn74And2 USE ENTITY WORK.and2(TechSn74);
FOR ALL: dn84And2 USE ENTITY WORK.and2(TechDn84);
```

In the above case, the same symbol (*and2*) is used for all instances of the component.

For all resource libraries visible through the library clause, VHDL In creates a property on the generated schematic with the following attributes.

```
Property Name:vhdlLibraryNames
Property Type:ilList of strings
Property Value:(<library name>...)
```

Example property names for a resource library are `ieee` and `myTestLib`.

For all packages visible to the architecture, VHDL In creates a property on the generated schematic with the following attributes.

```
Property Name:vhdlPackageNames
Property Type:ilList of strings
Property Value:(<complete package name>...)
```

Example property names for a package are `ieee.std_logic_1164.all` and `myTestLib.package.all`.

Case Sensitivity Issues

The Cadence library structure and the OA format are case sensitive, whereas the VHDL library structure is case insensitive. The intermediate files generated by the parser do not preserve the case of the original declaration in OA format. Therefore, libraries that are recognized as distinct in the Cadence library structure are not distinguishable in VHDL.

Possible VHDL Design Import Errors

When you import a VHDL design, this difference in case sensitivity between the three formats might cause errors with design libraries, cells, or ports.

■ Design libraries

In a VHDL library, the design names `Example`, `EXAMPLE`, and `example` map to the same VHDL logical library, while in the Cadence library structure, each maps to a different Cadence library structure. This might cause an error if VHDL In preserves the original case of VHDL declarations.

■ Cells

In VHDL, components bound to the entities named `and` and `AND` are bound to same entity. In the Cadence library structure, the components are recognized as different cells.

■ Ports

In a VHDL design, the name declared in the original definition of the port in the entity declaration and the name used for the formal part in the port map clause might not be expressed as both uppercase or as both lowercase. This causes an error when the

design is imported to OA format. VHDL In maps actual ports to formal ports even if both names are not in the same case. The following VHDL example illustrates the point.

```
entity Example is
  port (a: in bit);
end entity;
INSTANCE: example port map (A =>actual_signal);
```

A search for port *A* in the symbol for entity Example succeeds in VHDL In while matching for symbol pins.

Object Search in OA

Object search in OA is case insensitive.

For example, the statement

```
FOR ALL: and2 USE ENTITY sn74.AND2_GATE (TECH_TTL);
```

performs a case-insensitive search in the `sn74` library for the `and2_gate` cell.

If you want to preserve the case of an identifier when searching a component symbol in a OA reference library, you can

- Use escape characters (`\\`), if you have specified binding for the component symbol you want to search
- Set a parameter in the parameter file

Case-Sensitive Search of an Identifier with Binding

If you have specified binding for the component symbol you want to search, you must surround the identifier with escape characters (`\\`).

For example, to make sure that VHDL In searches for the component `INV0` as `INV0` rather than `inv0` in the OA reference library, indicate the component as `\\INV0\\`.

Case-Sensitive Search of an Identifier with No Binding

Include the following statement in your parameter file if you want the component search to be case insensitive:

```
caseSensitivity := False
```

VHDL In will pick up the component from the OA reference library provided the component interfaces match.

This flag is `TRUE` by default.

Note: You can use this feature only to search for symbols. You cannot use this feature to

- Create symbols, entities, packages, configurations, or architectures
- Perform a case-sensitive search on an entire entity, package, configuration, or architecture

Repeated Instantiations of a Component

Once you have set the case-sensitivity flag to `FALSE` for the search of a component symbol, VHDL In uses the same process in searching for repeated instantiations of that component.

For example, if you specify the following instantiations for your search,

```
IO: IV120 PORT MAP(...  
I1: Iv120 PORT MAP(...
```

VHDL In searches the ordered list of symbols in the target and reference libraries for both `IV120` and `Iv120`, and returns the first case-insensitive match.

When neither instantiation `IV120` or `iv120` are present in the target library or the reference libraries, VHDL In creates a symbol for `IV120` in the target library and bind the component `IV120` to that symbol.

Library Names in All Uppercase

You must use escape characters (`\\`) to specify a library whose name is in all uppercase characters, even when you set the case-sensitive flag to `ON`.

For example, if the reference library `refLib` contains all the component symbols, you must use the following two statements:

```
LIBRARY \refLib\  
...  
...  
for GB3:IV120 use entity \refLib\IV120(schematic);
```

Process Involved in a Case-Sensitive Search

The following process occurs when you perform a case-sensitive search on a component with no binding:

- VHDL In generates an ordered list of symbols present in the library for the component you want to search.

- VHDL In searches the target library for the symbol you specified by traversing the list of symbols and comparing each symbol listed with the symbol you specified.
- If VHDL In does not find the symbol you specified in the target library, it searches the list of reference libraries, using the same process it used to search the target library.
- VHDL In performs a check on pin names corresponding to the component, searching first the target library, then the reference libraries.

VHDL In provides more flexibility in pin name searching than it does in component name searching.

For example, if you use the component declaration

```
component IV120
port(Y:out std_logic; a:in std_logic);
end component;
```

and later use the component instantiation statement

```
GB3: IV120 port map(Y => y, A => b);
```

VHDL In will successfully find a match for the port name if `IV120` is present in the reference library, and the ports for `IV120` are `Y` and `A`.

Object Creation in OA

Object creation in OA preserves the original case of the declaration.

For example, the code

```
ENTITY AndGate IS
    PORT (a: IN std_logic; B: OUT std_logic);
END AndGate;
```

creates a cell called `andgate` with ports called `a` and `b`.

Other Issues

Signal Declarations

VHDL In can create schematic representations for only these two types of signal declarations:

- Scalar
- A single dimensional array of scalar

For other types of signal declarations in VHDL, there is no mapping to the OpenAccess database (OA).

Note: In case of port bundles you can specify the VHDL datatype for individual signals as `sigName1:sigType1`, `sigName2:sigType2`, `sigName3:sigType3`. For example, you have defined two signals, `sig1` and `sig2` as, `vhdlDataType : sig1:std_logic`, then `sig1` has `std_logic` as the datatype and `sig2` has the default scalar datatype. If you have specified `vhdlDataType : std_logic`, both signals, `sig1` and `sig2`, have `std_logic` as the datatype. If you do not specify the `vhdlDataType` then both the signals have the default scalar datatype defined in the *VHDL Set Up Check* dialog box.

Vector Nets

The Cadence netlister cannot handle vector nets which are constrained to a single dimension, such as the following:

```
SUBTYPE constrained_singleDimension IS bit_vector(0 TO 9);
```

In a schematic, a vector net `B<0:9>` of type `constrained_singleDimension` is netlisted as

```
SIGNAL B: constrained_singleDimension(0 TO 9);
```

which is not correct VHDL. VHDL In adds a property with the following definition to indicate VHDL data type:

```
Property Name: vhdlDataType  
Property Type: string  
Property Value: <dataType>
```

An example VHDL data type is `std_logic`.

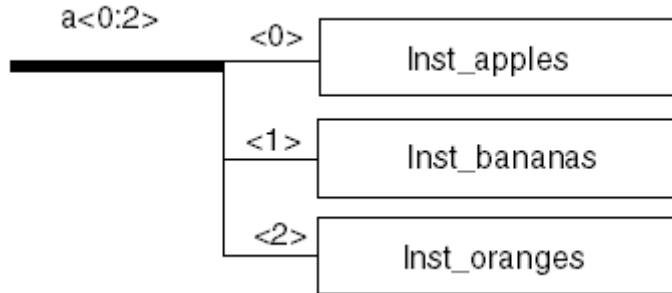
Noninteger Vector Indices

The following is an example of a `vector` type with an index subtype other than `integer`.

```
PACKAGE td IS  
    TYPE foo is (apples, bananas, oranges, peaches);  
    TYPE index_foo_vector is array (foo range <>) of bit;  
END td;  
  
USE work.td.all  
  
ENTITY e IS  
    PORT (a: IN index_foo_vector (apples to oranges));  
END e;  
  
ARCHITECTURE a OF e IS  
    COMPONENT foo  
    PORT (a: in bit);  
    END component;
```

```
BEGIN
  inst_apples: foo PORT MAP(a (apples));
  inst_bananas: foo PORT MAP(a (bananas));
  inst_oranges: foo PORT MAP(a (oranges));
END a;
```

Importing the example produces a schematic with the following connectivity.



VHDL In creates a property on net *a* with these characteristics:

Property Name: `vhdlVecIndexType`
Property Type: `string`
Property Value: `<index_type>`

An example index type is `libPack.td.foo`.

Buffer Ports

VHDL In maps buffer ports in VHDL to INOUT ports on Virtuoso schematics and symbols and adds the following property.

Property Name: `vhdlPortType`
Property Type: `string`
Property Value: `<portType>`

An example port type is `buffer`.

Port Conversion Functions

For port conversion functions, VHDL In creates the following properties on the instance terminal.

For conversion applied on the formal port

Property Name: `vhdlFormalPortFuncName`
Property Type: `string`
Property Value: `<function name>`

An example function name is `bitToMv1` for the port map `(bitToMv1 (A) => Net A, ...`

For conversion applied on actual port:

```
Property Name: vhdlActualPortFuncName
Property Type: string
Property Value: <function name>
```

An example function name for an actual port is `bitToMv1` for the port map `(A => bitToMv1(Net A), ...`

Attribute Specifications

Schematic generation deals with attribute specification on an object in an architecture by creating the following property on the corresponding object in the schematic.

```
Property Name: vhdlAttributeDefList
Property Type: ilList of strings
Property Value: (<attribute name>, <attribute value>)
```

For example, the property for the specification `((Capacitance \ 15.2F\))` on net `net21` having the attribute `Capacitance` of `net21: signal is 15.2F`.

Importing Generics

A generic clause declared in a VHDL entity is mapped as a property called `vhdlGenericDefList` on the symbol. The syntax of such a property is:

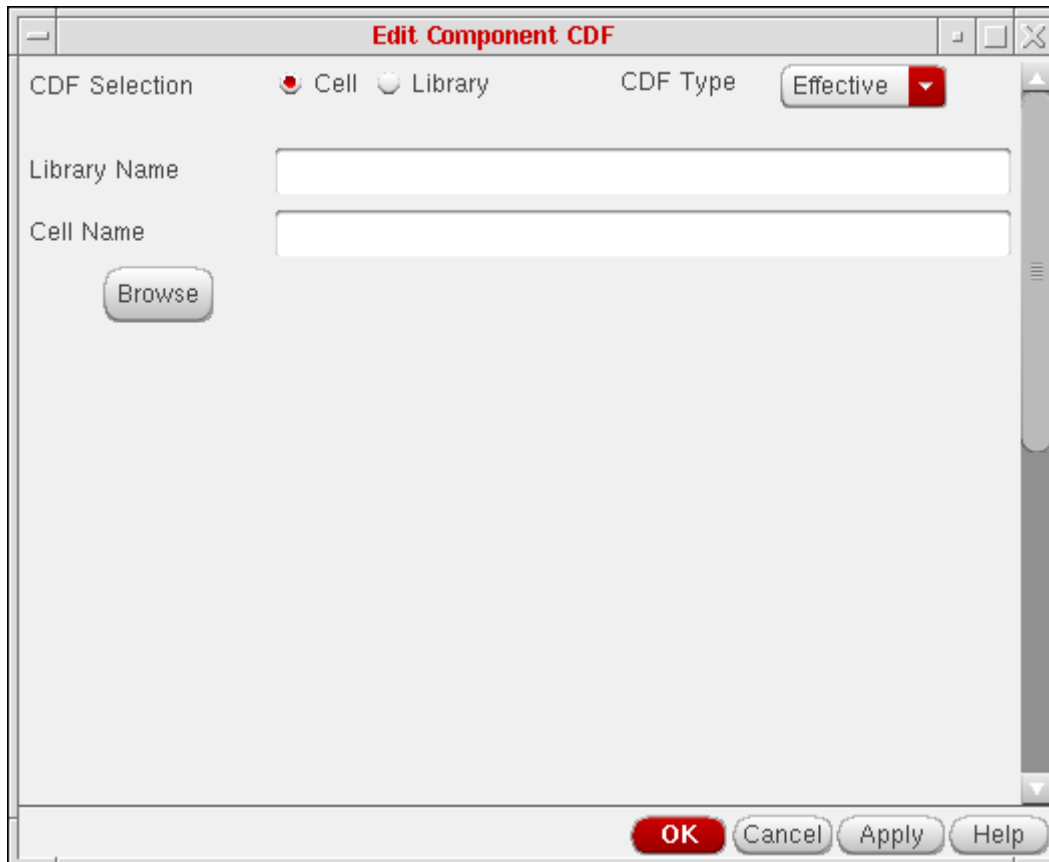
```
Property Name: vhdlGenericDefList
Property Type: ilList
Property Value: ((<name><type>[<value>])...)
```

Example clauses are `(delay, time)` and `(clock frequency, frequency, 45MHz)`.

A generic map clause in the component instantiation statement is also mapped as `vhdlGenericMapList` on the component instance in the schematic.

In addition to this, a generic clause declared in a VHDL entity is also translated to a Component Description Format (CDF) Parameter. The generics can be viewed through the Edit Component CDF form in `icds`. To display the Edit Component CDF form, select the *Edit* option from the *CDF* sub-menu in the *Tools* drop down menu.

The Edit Component CDF form displays the various options for displaying the generics. You



can choose the CDF Selection default value, Cell, and specify the Library Name and the cell name of the cell which contains the generic clause. Next, you select the CDF Type to be Base.

This will display any generics specified in the VHDL entity description of the cell with their default values.

The cell CDF parameters are inherited by the corresponding component instantiations as CDF Properties. Therefore they can be viewed by selecting the component in the schematic opened through the Library Manager and selecting the Edit Properties button.

In case the component instance has a generic map clause giving its own value for the generic, this value overrides the value inherited from the cell.

Component Declarations

In VHDL, a component declaration can exist in an architecture. It has to be present in one of the referred packages or architecture declarations. It is an error in VHDL to have the same

component declared in two places. The VHDL netlister does not support such cases. VHDL In adds the following property for instances of components whose declaration does not exist inside an architecture.

```
Property Name: vhdlPackageComponents  
Property Type: ilList of strings  
Property Value: (<componentName>...)
```

Example component names are And2Gate and Nand2Gate.

Port Maps

If a port map other than a simple port map is present on a binding specification, VHDL In imports the module as a VHDL view. This is necessary because the port map on the binding specification has no relevant abstraction in OA.

The following binding specification is imported as a schematic.

```
for all: and2 use entity CellsLib.And2(Structure) port map  
  (A => A, B => B, Y => Y);
```

The following binding specification is imported as a VHDL view.

```
for all: and2 use entity CellsLib.MyAnd2(Structure2) port  
  map(Input1 => A, Input2 => B, Output => Y);
```

VHDL In supports port maps on the component instance statement.

Concurrent Assignment Statements

VHDL In supports only simple concurrent assignment statements on the schematic view. A simple concurrent assignment has only one waveform element and no timing expression, or the right-hand side (RHS) of the assignment is an assignable value. For example,

Assignments that are simple

```
a <= b(3 to 6); a <= b;
```

Assignments that are not simple

```
a <= b after 5 ns; c <= function (d);
```

For imported simple concurrent assignment statements, VHDL In uses an instance of a patch symbol. The patch expression for such an instance is

```
Connection Expression 0:size-1 0: size-1.
```

VHDL In adds the following property to the patch instance.

Property Name: `vhdlAssignInstance`
Property Type: `Boolean`
Property Value: `<true/false>`

An example property value is `TRUE`.

VHDL In also supports signal concatenation. For example, if a concurrent statement is

```
selectx <= Sel(2) & Sel(1) & Sel(0);
```

VHDL In represents this assignment as three separate bit-by-bit assignments.

Behavioral View for Continuous Signal Assignment Statements

To create a schematic view, VHDL In requires the patch cord symbol from the `basic` library if the design contains Continuous Signal Assignment statements. It creates a behavioral view if any one of the following conditions is true:

- The `basic` library is not defined in the `cds.lib`.
- The cell `patch` view symbol is not present in the `basic` library.
- The value of the Parameter File entry for `contAssignSymbol` is not `basic patch` symbol, and the patch symbol is not available in the desired library. This is applicable if you are running VHDL In in stand-alone mode.

The log file displays the relevant messages.

Power and Ground Signals

In VHDL designs, power and ground nets are represented as continuous assignments where the left-hand side (LHS) is an enumeration literal designated as power or ground. For example,

```
a <= '1'
```

indicates that net `a` is a power net, and

```
b <= '0'
```

indicates that net `b` is a ground net.

You need to supply the enumeration literals designated for power and ground.

Nontranslatable Structural VHDL Constructs

Some elements of VHDL simply cannot be converted into schematics or netlists. If you run VHDL In on a design that contains one or more of these elements, VHDL In automatically switches the output to a Cadence VHDL text cellview and issues warnings that other conversions are not possible.

The following section describes the structural VHDL constructs that VHDL In cannot convert into OA objects. Check your design for these constructs, and try to change them or remove them before importing the design.

Port Subtypes and Modes

VHDL In cannot convert the following port subtypes:

- Globally static arrays
- Unconstrained arrays
- Arrays of composite objects
- Record types
- Multidimensional array of scalar objects

VHDL ports have five modes: `in`, `out`, `inout`, `linkage`, and `buffer`, while OA allows three port modes: `in`, `out`, and `inout`.

Ports of mode `in`, `out` and `inout` in VHDL are directly mappable to corresponding ports in OA.

VHDL In cannot convert ports of mode `linkage`.

In VHDL, you can associate ports of mode `buffer` with only a signal or with only ports of mode `buffer`. In VHDL, modes `inout` and `buffer` have the same characteristics, but you can update ports of mode `inout` with zero or more sources, while you can update ports of mode `buffer` with at most one source. Therefore, VHDL In maps ports of mode `buffer` to ports of mode `inout` in OA and associates properties with them to indicate that these are `buffer` ports.

In VHDL, you can associate ports of mode `linkage` with ports of any mode. However, the value of the interface object can be read or updated only by appearing as an actual to an interface object of mode `linkage`, and no other reading or updating of the value is permitted. VHDL In cannot map ports of mode `linkage` to the modes of a OA port.

The following VHDL example shows constructs that are problematic for VHDL In and describes their resolution.

```
type multi_dim_array is array(1 to 10, 1 to 10) of bit;
type rec_type is record scalar_element: bit; vector_element:
    multi_dim_array; end record;
type sig_dim_array is array (integer range <>) of bit;
type comp_sig_dim_array is array (1 to 10) of rec_type;
type const_single_dim_array is array (1 to 10) of bit;

entity ISSUE is
port (a: in multi_dim_array;
      b: out rec_type;
      c: inout sig_dim_array(1 to 10);
      d: in sig_dim_array;
      e: buffer comp_sig_dim_array;
      f: linkage const_single_dim_array;
      g: integer range 1 to 10);
end ISSUE;
```

- | | |
|---------------|---|
| Port a | The port is a multidimensional array of <code>scalar</code> type. OA ports can be either scalar or constrained single dimensional arrays of scalars. This port has no direct mapping to any OA object. Therefore, VHDL In cannot convert this port to a OA symbol. |
| Port b | The port is a <code>record</code> type. In VHDL, different elements of a record can have different subtypes. These subtypes can be a composite. VHDL In cannot create this port in a OA symbol. |
| Port c | The port is a single-dimensional array of <code>scalar</code> type. The base type of the array is an <code>unconstrained</code> type. This port declaration creates an <code>anonymous</code> type. VHDL In can create this port in OA. Although the declarations for ports <code>f</code> and <code>c</code> are <code>vector</code> of size 10, the declaration for port <code>c</code> generates an <code>anonymous</code> type implicitly while the port <code>f</code> declaration does not. |
| Port d | The port is an unconstrained single dimensional array. This single dimensional array port gets the value of its constraints from the associated signal in each instance. Because OA has no parameterized ports, VHDL In cannot convert this port into a OA symbol. |
| Port e | The port is an array of a <code>composite</code> type (arrays and records). Different <code>scalar</code> elements of an array of <code>composite</code> type can be of different types. VHDL In cannot map such a port to a OA port. |
| Port f | The port is of mode <code>linkage</code> , which cannot be mapped to OA ports. |

Port g The port is a `scalar` type. This declaration implicitly creates an `anonymous` type. VHDL In can convert this port to a OA symbol.

Constructs in Architectures

VHDL In cannot convert the following VHDL constructs that occur in an architecture body.

- Binding specifications other than simple ones
- Disconnection specifications
- Signal declarations of these types:
 - Arrays of composite type
 - Records
 - Declarations that are not locally static
- Concurrent versions of these statements:
 - Process statements
 - Procedure calls
 - Assertion statements
 - Complex signal assignment statements
 - Generate statements
 - Blocks with guard expressions
- Primary units with inconvertible entities

Support of concurrent assignment statements is restricted to simple concurrent assignment statements. A simple signal assignment statement generated by a synthesis tool has only one waveform element. The value expression of the waveform element is a simple or selected name. The value expression cannot contain operators, function calls, or function- or signal-valued attributes.

Expressions associated with the formal part can be signal-valued attributes. Signal-valued attributes do not have an explicit place holder for the declaration of the signal. This signal is related to the signal prefix of the attribute. Signal-valued attributes can occur recursively. OA does not define such relationships between nets. VHDL In cannot generate a schematic view if signal-valued attributes are associated with the `formal_part` in a port map clause.

Blocks are logical partitions in VHDL design. Multiple sheet schematics in OA represent mapping to blocks. Because OA sheets have a finite size, a block might not fit in one sheet. The presence of guarded expressions on the block causes an implicit declaration of a signal named `guard`. This signal might control the operation of a concurrent signal assignment statement. VHDL In cannot map this behavior to objects in a OA schematic.

Error Messages

This chapter discusses the following:

- Dialog Boxes
- Text String Error Messages
- Using the VHDL Tool Box Status Window

Dialog Boxes

As you use the VHDL Import form, various situations cause a dialog box to open. Each is the result of a different problem or situation. If you enter an invalid string in a text field, VHDL In highlights the box with a flashing border. The flashing border goes away when you enter a valid string.

In general, when a dialog box opens, clicking *OK* or *Cancel* on the VHDL Import form closes the box.

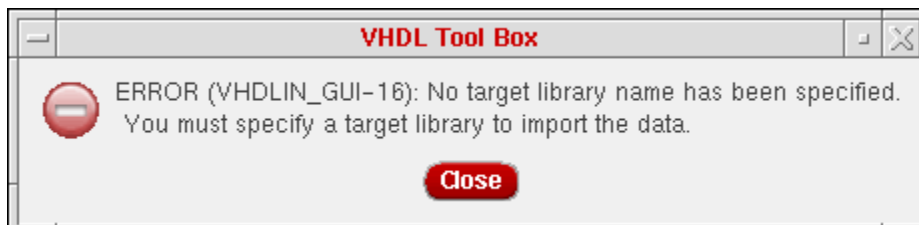
File Name Does Not Exist

If you type a string into the *File Name* field that does not match an existing file or directory, or does not include any wild card characters, the following error message appears.



No Target Library Name Specified

If you click either *OK* or *Apply* on the VHDL Import form without filling in the *Target Library Name* field, the following error message appears.



Library Name Not Found

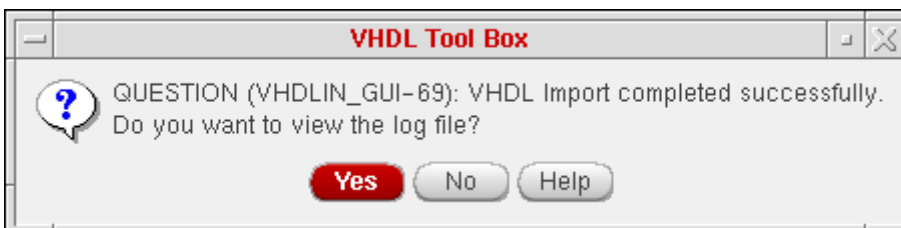
If you enter a target library name that does not exist in the *Target Library Name* field, the following dialog box opens. It asks you if you want to create the library, or if you mistyped the

library name. Clicking *OK* opens the Create Library form, which asks you for the location of the library.



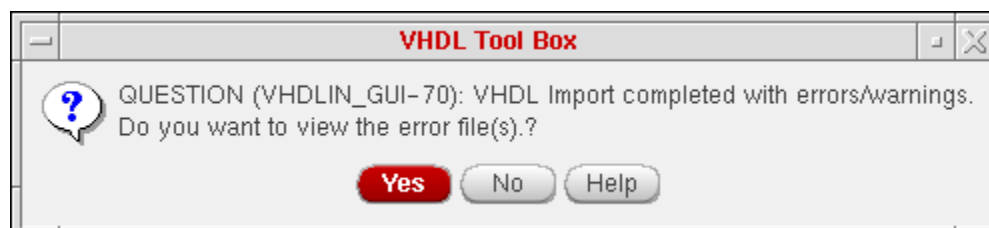
Import Completed Successfully

When the import process finishes successfully, the following dialog box opens. If you click *OK*, the VHDL Tool Box status window opens, displaying the contents of the log file.



Import Completed with Errors/Warnings

When the import process finishes with errors or warnings, the following dialog box opens. If you click *OK*, the VHDL Tool Box status window opens, displaying the contents of the first error or warning file. You can also view the log file from this status window.



Import Aborted When Reached Maximum Errors

If the import process reaches the maximum number of errors, the process ends and the following dialog box opens. If you click *OK*, the VHDL Tool Box status window opens,

displaying the contents of the first error or warning file. You can also view the log file from this status window.



Text String Error Messages

The following section summarizes the different text string error messages that you might receive while using VHDL In.

Overview

All text string error messages have the following syntax:

```
vhdlin: *<severityCode>,<errorNumber> <errMsgString>\n
        [error Causing line] [(errorFileName), (lineNumber)]
```

The *severityCode* can be *F*, *W*, or *C*.

F indicates a fatal error, *W* indicates a warning, and *C* indicates a fatal error that allows you to continue processing. The import process finishes without importing the design on the following two conditions:

- On encountering the first fatal error
- If there is at least one error of severity *C* after analysis is finished

Line Command Errors

If you are using the standalone line command, and you type in one or more unknown or ambiguous command line options or functions, you receive an error message. The unknown or ambiguous option words are shown in parentheses. If the option is ambiguous, you must add additional characters to select a single option. Examples of command line error messages are

```
vhdlin: *F,5: unknown or ambiguous options (%s)
vhdlin: *F,5: Missing -param option, -param <parameter_filename> not given while
invoking the tool.
```

VHDL In for Virtuoso Design Environment User Guide and Reference

Error Messages

```
vhdlin: *F,5: -PARAM option must be followed by parameter file name.
```

You can use the *-help* option to list all valid options.

Analyzer Errors

If the import process is unable to fork the VHDL analyzer, or if the VHDL analyzer terminates with an internal error, you might receive error messages like the following.

```
vhdlin: *F,5: cannot execute the VHDL analyzer (%s)
vhdlin: *F,5: VHDL analyzer execution error
```

If the import process is unable to run the VHDL analyzer on the design files that you want to import, then you might need to set your UNIX search path to include the location of the VHDL analyzer. The UNIX file name for the VHDL analyzer is indicated in the parentheses.

If you encounter an analyzer internal error, contact Cadence Customer Support.

Parameter File Parsing

A problem in the parameter file, such as an illegal parameter or an illegal parameter value, can cause a variety of error messages.

```
vhdlin: *F,5: invalid parameter specification (%s) in the parameter file
vhdlin: *F,5: %s argument(s) is/are expected for the parameter (%s)"
vhdlin: *F,5: invalid value (%s) specified for parameter (%s)"
vhdlin: *F,5: Parameter (%s) is expected but not specified in the parameter file
vhdlin: *F,5: parameter (%s) is required to have only integer value"
vhdlin: *F,5: parameter (%s) has valid integer range (%s), specified value lies
outside this range"
vhdlin: *F,5: parameter (%s) is required to have only real value"
vhdlin: *F,5: parameter (%s) has valid real range (%s), specified value lies outside
this range"
vhdlin: *F,5: parameter (%s) has valid values as (%s), specified value lies outside
this range"
vhdlin: *W,5: parameter (%s) not in use"
```

Correct the parameter specification, and rerun the import process.

Virtuoso Design Library

If the system cannot correctly resolve references to the Virtuoso design library, it produces an error message similar to one of the following.

VHDL In for Virtuoso Design Environment User Guide and Reference Error Messages

```
vhdlin: *W,5: Could not find view of type (%s) for the cell (%s) in the library (%s)
vhdlin: *W,5: Unable to open library %s in %s mode."
vhdlin: *W,5: Unable to find View with name %s in the library %s.
```

Memory

If no more memory is available on your workstation, or the memory is corrupted, the import process ends and prints one of these messages:

```
vhdlin: *F,5: cannot allocate more memory
vhdlin: *F,5: Memory Allocation corrupted = %d.
vhdlin: *F,5: Dynamic Memory corrupted = %d.
```

Consult your system administrator to increase the size of memory on your machine. The import process should not corrupt your memory. If this happens, call Cadence Customer Support.

General Errors

If the system cannot open a file specified for reading or writing by the import process, you see this message.

```
vhdlin: *F,5: cannot open specified file (%s) for %sing.
```

Use the correct file or change access permissions on the file.

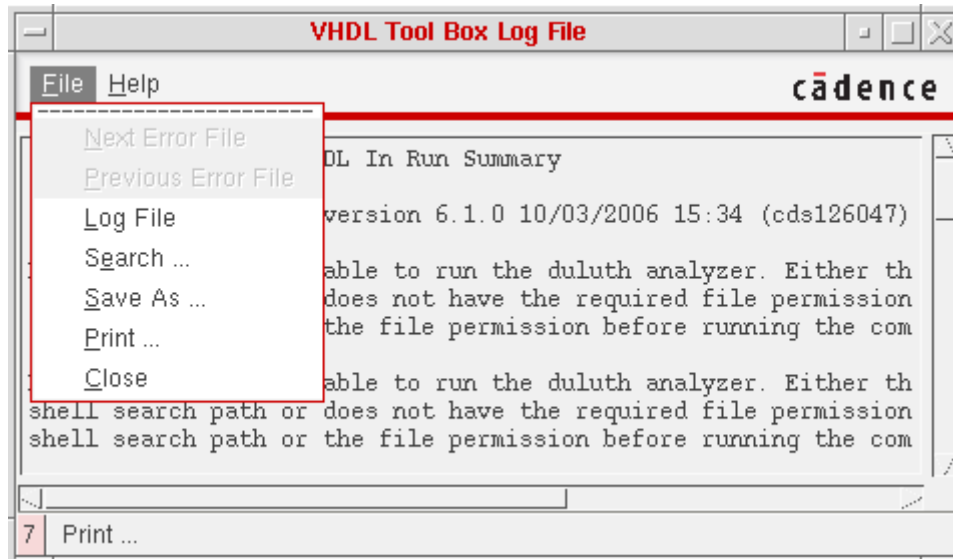
Using the VHDL Tool Box Status Window

Although the examples in this user guide have you only look at and close the VHDL Tool Box status window, the window has a set of commands that can help you debug problems or understand how your design is imported.

VHDL In for Virtuoso Design Environment User Guide and Reference

Error Messages

Whether you are looking at an error file or a log file, you can use the same menu of commands under *File* in the window menu banner.



You have the choice of the following commands.

- *Next Error File*
- *Previous Error File*
- *Log File*
- *Search*
- *Save As*
- *Print*
- *Close*

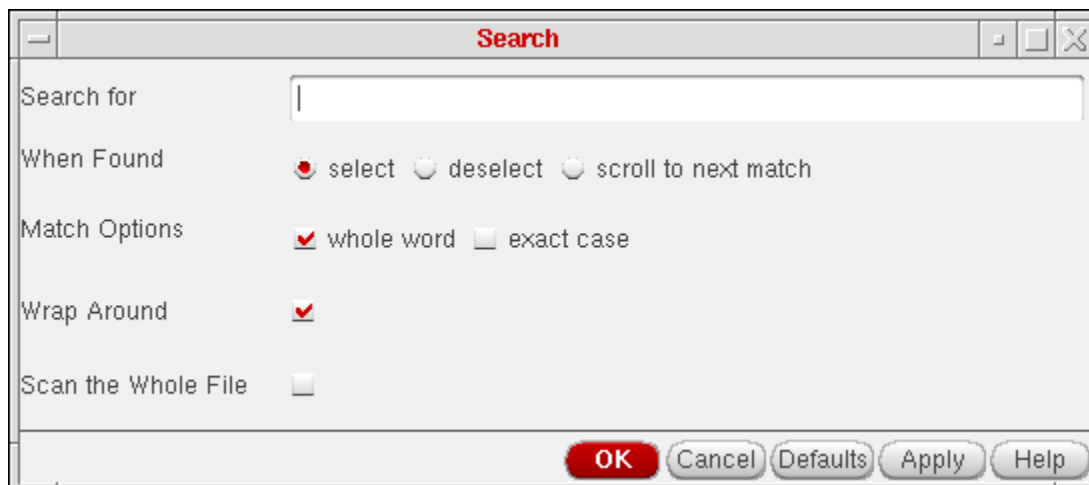
The first two commands are not available (shaded out) if you are looking at the log file of a successful importation, or if you are looking at the only error file.

Next Error File takes you to the next error file. If you are viewing a log file, this command takes you to the corresponding error file for the same design.

Previous Error File takes you to the previous error file.

Log File does nothing if you are already looking at the log file. This command takes you to the log file if you are looking at the error file.

Search opens the following form, which prompts you for a character string to search for.



The screenshot shows a dialog box titled "Search" with a standard Windows-style title bar (minimize, maximize, close buttons). The dialog contains the following elements:

- A "Search for" label followed by a text input field.
- A "When Found" section with three radio buttons: "select" (selected), "deselect", and "scroll to next match".
- A "Match Options" section with two checkboxes: "whole word" (checked) and "exact case" (unchecked).
- A "Wrap Around" checkbox (checked).
- A "Scan the Whole File" checkbox (unchecked).
- A row of five buttons at the bottom: "OK" (highlighted in red), "Cancel", "Defaults", "Apply", and "Help".

Save As opens the Save As form, which lets you type in a file name for saving the current data.

Print opens the Print form, which lets you print the error or log file.

Close closes the Errors/Warnings list window.

The most common errors are caused by the wrong combination of design files selected for importation. Typically, a critical file is missing, or you imported too many files.

Creating a Parameter File

This chapter discusses the following:

- Parameter File Structure
- Description of Parameters

Parameter File Structure

This chapter describes the statements you must use when you create a VHDL In parameter file. You can use this file when running VHDL In in standalone mode. A VHDL In parameter file must consist of a series of single line entries, each with the following syntax:

```
parameter_name := parameter_value -- <comments>
```

Here is a long parameter file. This file supports the conversion of a VHDL adder design into a Virtuoso schematic.

```
logFileName := /tmp/vhdlImLob14298
errorFileName := /tmp/vhdlInErb14298
importLibraryName := adder
inputPinSymbol := basic ipin symbol
outputPinSymbol := basic opin symbol
inoutPinSymbol := basic iopin symbol
contAssignSymbol := basic patch symbol
ignoreContAssFunc := FALSE
writeInRefLibs := TRUE.
structuralViewType := schematic
referenceLibraries := basic,US_8ths
symbolViewName := symbol
overwriteExistingView := TRUE
maxError := 10
powerType := std_ulogic
powerLiterals := '1'
powerNetName := VDD!
groundType := std_ulogic
groundLiterals := '0'
groundNetName := GND!
sheetBorderSize := US_8ths Asize symbol
maxNoRows := 1024
maxNoCols := 1024
fontHeight := 0.062500
lineLineSpacing := 0.200000
lineComponentSpacing := 0.500000
componentDensity := 0
pinPlacement := left_right_boundaries_only
fullPlaceRouteSchematic := TRUE
squareSchematics := TRUE
minimizeCrossovers := TRUE
optimizeLabels := TRUE
caseSensitivity := FALSE
generateFastSchematic := TRUE
fastSchematicMaxInst := 20000
fastSchematicMaxPort := 5000
```

Description of Parameters

The following sections describe each parameter and give a sample parameter specification. If you leave out a parameter, the system assigns it the default.

Anything that is not an integer or real number is a string. A string does not need quotes. Trailing and preceding white spaces have no effect. Blank spaces inside strings are preserved. You can write only one parameter specification per line. You can continue a specification onto a second line by using a backslash (\) at the end of the first line.

logFileName. UNIX file where VHDL In places the log of the import process. The default is `vhdlin.log`.

```
logFileName := vhdlin.log
```

errorFileName. UNIX file where VHDL In places any error messages that occur during the import process. The default is `vhdlin.err`.

```
errorFileName := vhdlin.err
```

importLibraryName. Specifies the logical library in OA format where VHDL In places the output. If you do not provide a path to this library in your `cds.lib` file, and if the directory does not exist, then VHDL In creates an import library in the current working directory and appends its definition in the `cds.lib` file. There is no default. You must always provide a library name.

```
importLibraryName := test_library
```

inputPinSymbol. Specifies the symbol that VHDL In must instantiate for input pins. If you do not specify a symbol, or VHDL In cannot find it, VHDL In cannot generate a schematic. There is no default. You must always provide a value if you are trying to generate structural netlist or schematic.

```
inputPinSymbol := basic ipin symbol
```

outputPinSymbol. Specifies the symbol that VHDL In must instantiate for output pins. If you do not specify a symbol, or VHDL In cannot find it, VHDL In cannot generate a schematic. There is no default. You must always provide a value if you are trying to generate a structural netlist or schematic.

```
outputPinSymbol := basic opin symbol
```

inoutPinSymbol. Specifies the symbol that VHDL In must instantiate for inout pins. If you do not specify a symbol, or VHDL In cannot find it, VHDL In cannot generate a schematic. There is no default. You must always provide a value if you are trying to generate a structural netlist or schematic.

```
inoutPinSymbol := basic iopin symbol
```

contAssignSymbol. Specifies the symbol that needs to be instantiated for a continuous assign statement. If such a symbol is not present, then VHDL In cannot generate a schematic. There is no default. You must always provide a value if you are trying to generate a structural netlist or schematic.

```
contAssignSymbol := basic patch symbol
```

ignoreContassFunc. Ignore continuous assignment statement function. Some continuous assignment statements assign the function of a variable to another variable, such as `a <= f(b)`. This is impossible to netlist and must be treated as a behavioral statement. If `ignoreContassFunc` is `true`, VHDL In ignores the function `f()`, treating the statement as a simple assignment. Use this parameter with caution, because setting it to `TRUE` can change a statement such as `a <= not(b)` so that `a` is aliased to `b`, which is not correct.

```
ignoreContassFunc := TRUE
```

writeInRefLibs. If this parameter is set to `true`, VHDL In creates symbols in reference libraries when the component instances in the architecture are bound libraries that do not have the relevant symbol view.

```
writeInRefLibs := TRUE
```

structuralViewType. Valid view types are

- `vhdl`
- `schematic`
- `netlist`

The default is `schematic`.

```
structuralViewType := schematic
```

Corresponds to the Import Structural Architectures As Cyclic field on the VHDL Import form.

referenceLibraries. A list of libraries that VHDL In searches for the symbols of any unbound instances in an architecture. If VHDL In cannot find the symbol in any reference library, it creates a symbol in the destination library and instantiates it. The default is the import library, which VHDL In always searches first. If you provide other libraries, VHDL In searches them in order.

```
referenceLibraries := sample, basic
```

Corresponds to the Reference Libraries field on the VHDL Import form.

symbolViewName. The view name where the symbol file resides. The default view name is `symbol`.

```
symbolViewName := symbol
```

Corresponds to the Symbol View Name field on the VHDL Import form.

overwriteExistingView. This option has only two valid values: `TRUE` and `FALSE` (in uppercase or lowercase). When this parameter is `TRUE`, VHDL In overwrites the existing view in the destination library. The default is `TRUE`.

```
overwriteExistingView := TRUE
```

Corresponds to the Overwrite Existing Views click box on the VHDL Import form.

overwriteSymbolView. This option specifies whether to overwrite symbol of a module that already exists in the target library with the symbol being imported by VHDL In. This parameter has the following four valid values:

- 0 does not overwrite any symbol.
- 1 overwrites symbols created by VHDL In (symbols that have the `createdBy` property set as `vhdlin`).
- 2 overwrites symbols created by the Text-to-Symbol generator or any other tool (symbols that have the `createdBy` property set as any value other than `vhdlin`).
- 3 overwrites all symbols.

The default is 0.

```
overwriteSymbolView := 1
```

Corresponds to the Overwrite Symbol Views list box on the VHDL Import form.

maxError. An integer that specifies the maximum number of analysis errors per file that VHDL In reports in case the file analysis fails. The default is 10.

```
maxError := 10
```

Corresponds to the Maximum Number of Errors field on the VHDL Import form.

powerType. This option indicates the base type of the enumeration literal. The default is `std_ulogic`.

```
powerType := std_ulogic
```

Corresponds to the (Power) Data Type field in the Power section on the VHDL Import form.

powerLiterals. This option indicates which literal values on the LHS of a continuous assignment statement specify a power net. The default is '1'.

```
powerLiterals := '1', 'H'
```

Corresponds to the (Power) Value field in the Power section on the VHDL Import form.

powerNetName. Used in OA to indicate a power net. If you forget to put an exclamation point (!) at the end of the name, VHDL In automatically adds one. The default is "VDD!"

```
powerNetName := "VDD!"
```

Corresponds to the (Power) Net Name field in the Power section on the VHDL Import form.

groundType. This option indicates the base type of the enumeration literal. The default is *std_ulogic*.

```
groundType := std_ulogic
```

Corresponds to the (Ground) Data Type field in the Ground Section on the VHDL Import form.

groundLiterals. You can use this option a second time to indicate which literal values on the left-hand side (LHS) of a continuous assignment statement specify a ground net. The default is '0'.

```
groundLiterals := '0', 'L'
```

Corresponds to the (Ground) Value field in the Ground section on the VHDL Import form.

groundNetName. You can specify this parameter a second time to indicate a ground net in OA. If you forget to put an exclamation point (!) at the end of the name, VHDL In automatically adds one. The default is "GND!"

```
groundNetName := "GND!"
```

Corresponds to the (Ground) Net Name field in the Ground section on the VHDL Import form.

sheetBorderSize. The frame symbol used for the sheet border. There is no default. If you do not define this parameter, no order is instantiated.

```
sheetBorderSize := US_8ths Asize symbol
```

Corresponds to the Sheet Border Size cyclic field under the *Schematic Generation Options* tab.

maxNoRows. An integer that specifies the number of components that VHDL In can place in one row of a schematic page up to a maximum of 1024. This parameter is required only if you specify a sheet border. The default is 1024.

```
maxNoRows := 1024
```

Corresponds to the Maximum Number of Rows field under the *Schematic Generation Options* tab.

maxNoCols. An integer that specifies the number of components that you can place in one column of a schematic page. This parameter is required only if you specify a sheet border. The default is 1024.

```
maxNoCols := 1024
```

Corresponds to the Maximum Number of Columns field under the *Schematic Generation Options* tab.

fontHeight. A real number that specifies the height of wire labels in user units. You must use Virtuoso Schematic Editor L to specify which user units you are using. Pin labels are 75% of the specified height. The default is 0.0625.

```
fontHeight := 0.0625
```

Corresponds to the *Font Height* field under the *Schematic Generation Options* tab.

lineLineSpacing. A real number, representing the space, in inches, between two adjacent nets. The range is 0.125 to 0.625 inches.

```
lineLineSpacing := 0.2
```

Corresponds to the *Line to Line Spacing* field under the *Schematic Generation Options* tab.

lineComponentSpacing. A real number, representing the space, in inches, between a component and an adjacent net. The range is 0.125 to 0.625 inches.

```
lineComponentSpacing := 0.125
```

Corresponds to the *Line to Component Spacing* field under the *Schematic Generation Options* tab.

componentDensity. An integer that controls the number of instances on one page of a schematic with a sheet border. The range is 0 to 100 instances.

```
componentDensity := 0
```

Corresponds to the *Component Density* sliding field under the *Schematic Generation Options* tab.

pinPlacement. Specifies the pin placement for the schematic view. Valid options are

- *left_right_boundaries_only*
- *all_boundaries*
- *file, pinPlacementFileName*

```
pinPlacement := all_boundaries
```

The default is *left_right_boundaries_only*. If you specify *file*, VHDL In expects a second argument that specifies the file name, following this syntax:

```
pinPlacement = file, <pinPlacementFileName>
```

The pin placement file must consist of single-line statements that use the following syntax:

```
componentPinPlacement ::= entityName, dir, pin_name_list
```

where

```
dir = top | bottom | left | right
```

and

```
pin_name_list = <pinName> [, <pinName>]
```

An example of a pin placement file is

```
componentPinPlacement := DFF, top, CLOCK
componentPinPlacement := DFF, left, D
componentPinPlacement := DFF, right, Q, Qn
```

Corresponds to the *Pin Placement* section under the *Schematic Generation Options* tab. This section has the radio buttons Left and Right Sides, All Sides, and Pin Placement File.

fullPlaceRouteSchematic. When you set this parameter to `FALSE`, VHDL In generates a schematic that is connected only by name, resulting in a faster creation of the schematic view and lower memory consumption. For an exceptionally large design, this gain is significant. The default is `TRUE`.

```
fullPlaceRouteSchematic := TRUE
```

Corresponds to the *Full Place and Route* click box under the *Schematic Generation Options* tab.

squareSchematics. If this parameter is `TRUE`, VHDL In performs squaring on schematics to break disproportionately long columns of components into many columns. The default is `TRUE`.

```
squareSchematics := TRUE
```

Corresponds to the *Generate Square Schematics* click box under the *Schematic Generation Options* tab.

minimizeCrossovers. If you set this parameter to `TRUE`, VHDL tries to enhance the aesthetic quality of the schematic by minimizing net crossovers. This slows down the schematic generation. For large designs, the slowdown is noticeable, while the improvement in aesthetics might not be so evident. The default is `TRUE`.

```
minimizeCrossovers := TRUE
```

Corresponds to the *Minimize Crossovers* click box under the *Schematic Generation Options* tab.

optimizeLabels. If you set this parameter to `FALSE`, VHDL In places net labels more rapidly. This could result in overlapping label names. The default is `TRUE`.

```
optimizeLabels := TRUE
```

Corresponds to the *Optimize Wire Label Locations* click box under the *Schematic Generation Options* tab.

caseSensitivity. If you set this parameter to `FALSE`, VHDL In performs a case-insensitive search of a component symbol in a reference library. The default is `TRUE`.

```
caseSensitivity := TRUE
```

The use of this parameter has certain restrictions. For more detailed information on the issues related to case-sensitive search, see [Chapter 1, “Conversion Issues.”](#)

generateFastSchematic. If set to `TRUE`, this parameter enables fast generation of the schematic when the design being imported contains a large number of instances or ports, specified in the `fastSchematicMaxInst` and `fastSchematicMaxPort` parameters.

```
generateFastSchematic := TRUE
```

For details on the `generateFastSchematic`, `fastSchematicMaxInst`, and `fastSchematicMaxPort` parameters, see [“Fast Schematic Generation”](#) on page 37.

fastSchematicMaxInst. Specify the number of instances. If the number of instances in the design being imported exceeds the specified number and `generateFastSchematic` is set to `TRUE`, VHDL In generates a schematic where the nets are not routed and the connectivity is indicated by names.

```
fastSchematicMaxInst := 20000
```

fastSchematicMaxPort. Specify the number of ports. If the number of ports in the design being imported exceeds the specified number and `generateFastSchematic` is set to `TRUE`, VHDL In generates a schematic where the nets are not routed and the connectivity is indicated by names.

```
caseSensitivity := TRUE
```

work_file. If you set this parameter to a library name, VHDL In will take this as the work library where the analyzed data will be created. The default is the target library.

```
work_file := my_work_lib
```

Corresponds to the *VHDL WORK Library Name* field under the *Schematic Generation Options* tab.

VHDL In Standalone Options

The chapter discusses the following:

- [Introduction](#)
- [Getting Help on a VHDL In Command](#)
- [Displaying the Version Number of VHDL In](#)
- [Suppressing Printing of the Copyright Banner](#)
- [Hiding the Display of Schematic Extraction Errors](#)
- [Compiling Functional Cellviews](#)
- [Specifying a cds.lib File](#)
- [Ignore Extra Pins on Symbols](#)
- [Specifying the VHDL WORK Library](#)
- [Specifying a Schematic Parameter File](#)
- [Specifying Multiple VHDL Source Files](#)
- [TDM and Imported VHDL Design Libraries](#)

Introduction

This appendix describes the options you can use in VHDL In standalone mode. You can use any of these options with additional options except the *-help* and *-version* options. Use the *-help* or *-version* option alone.

The syntax for using these options at a UNIX command prompt is

```
vhdlin -param param_file <option> <VHDL source files>
```

where

<code>vhdlin</code>	Specifies the command to start VHDL In in the standalone mode.
<code>-param</code>	Indicates that a <u>parameter</u> file will be read.
<code>param_file</code>	Specifies the vhdI import parameter file.
<code><option></code>	Specifies name of the option you want to use.
<code><VHDL source files></code>	Specifies the VHDL source files you want to import, using the options you specified.

Getting Help on a VHDL In Command

For more information on a VHDL In command, use the *-help* option.

The syntax is

```
vhdlin -help <VHDL In command>
```

where

vhdlin	Specifies the command to start VHDL In.
-help	Specifies the option you use to request help.

The output of this option is similar to the following:

```
@(#) $CDS: vhdlin version 4.4.1 11/22/96 09:44 (darbari) $: (c) Copyright 1994-1995. Cadence Design Systems, Inc.
```

```
Usage: vhdlin -param <paramFileName> -f <fileName> <vhdSourceFileNames>
```

Options are:

-HELP	--Prints this message
-VERSION	--Prints the version number
-NOCOPYRIGHT	--Suppress printing of copyright banner
+NOXTRSCH	--Do not extract
-CDSLIB <arg>	--Name of the cdslib file
-COMPILE	--Compile the functional cellviews
-WORK <arg>	--Specifies the VHDL WORK library
-SPEEDUP	--Speeds up VHDL In runtime; use only if entity source files precede architecture files.
-IGNOREEXTRAPINS	--Ignore extra pins to pick reference symbols
-PARAM <arg>	--Name of the schematic parameter file
-F <arg>	--Use to specify multiple VHDL source files.

Displaying the Version Number of VHDL In

When you want to the current version number of VHDL In displayed, use the *-version* option.

The syntax is

```
vhdlin -version
```

where

<code>-version</code>	Specifies the version of VHDL In available to you.
-----------------------	--

The output of this command is similar to the following:

```
@(#) $CDS: vhdlin_nc.exe version 6.1.2 09/26/2007 14:50 (cic612lnx)$
```

Suppressing Printing of the Copyright Banner

When you do not want to see the copyright banner, use the *-nocopyright* option.

The syntax is

```
vhdlin -nocopyright
```

An example use of this option is

```
vhdlin -nocopyright -help
```

Hiding the Display of Schematic Extraction Errors

If you do not want the schematic of your design to display extraction errors such as floating nets or shorted outputs, enter the *+noxtrs* option at the command prompt.

The syntax is

```
vhdlin -param param_file +noxtrs entity_file.vhd architecture_file.vhd
```

For example, to suppress the display of extraction errors when you import files for the full adder design (used in [Chapter 1, “Importing a Simple VHDL Design”](#)), use this command:

```
vhdlin -param_file -+noxtrs full_adder.e.vhd full_adder.a.vhd half_adder.e.vhd  
half_adder.a.vhd or_gate.e.vhd or_gate.a.vhd
```

Specifying a cds.lib File

When you want to specify a *cds.lib* file for VHDL In to read, use the *-cdslib* option.

The syntax is

```
vhdlin -cdslib <your cds.lib file>
```

An example use of this option is

```
vhdlin -cdslib <path>/cds.lib -param_file test.vhd
```

Compiling Functional Cellviews

When you want to compile functional (textual) cellview, use the *-compile* option.

The syntax is

```
vhdlin -compile
```

An example use of this command is

```
vhdlin -param param_file test.vhd -compile
```

Use this option to prepare functional cellviews for simulation. In this process, imported designs are analyzed in a target library (library of imported designs), whether or not the target library is managed by TDM.

Specifying the VHDL WORK Library

When you want to specify the name of the VHDL WORK library, use the *-work* option.

The syntax is

```
vhdlin -work <work library>
```

An example use of this option is

```
vhdlin -param param_file -work lib test.vhd
```

Ignore Extra Pins on Symbols

This controls the selection of symbols from the reference libraries. If you specify this and VHDL In finds a reference symbol with the same name as specified in the VHDL design, the symbol will be picked up. The pins not referred will remain unconnected in the schematic.

Note: The symbol will be picked up even if all its pins are not used.

Speeding Up Run Time

When you want to import several VHDL source files at one time for a large, complex design, you can speed up run time by using the *-speedUp* option. You must import the entity files before the architecture files.

The syntax is

```
vhdlin -param_file param_file -speedUp entity_file.vhd  
      architecture_file.vhd...
```

For example, if you want to import the following entity and architecture files for a top-level design:

```
block_entity.vhd  
block_arch.vhd  
top_entity.vhd  
top_arch.vhd
```

using the *-speedUp* option, use the command

```
vhdlin -param_file -speedUp block_entity.vhd block_arch.vhd top_entity.vhd  
      top_arch.vhd
```

where

block_entity.vhd	Specifies the <u>entity</u> file for the block design.
block_arch.vhd	Specifies the <u>architecture</u> file for the block design.
top_entity.vhd	Specifies the entity file for the top design.
top_arch.vhd	Specifies the architecture file for the top design.

Notice how each entity file is listed before its corresponding architecture file. This restriction is not applicable if you are running VHDL In without the *-speedUp* option.

Specifying a Schematic Parameter File

When you want to specify the name of a schematic parameter file, use the `-param` option.

The syntax is

```
vhdlin -param param_file
```

where

<code>param_file</code>	It is the schematic parameter file.
-------------------------	-------------------------------------

An example use of this option is

```
vhdlin -param param_file top_entity.vhd top_arch.vhd
```

where

<code>top_entity.vhd</code>	It is the entity file for the top design.
-----------------------------	---

<code>top_arch.vhd</code>	It is the architecture file for the top design.
---------------------------	---

Specifying Multiple VHDL Source Files

When you want to import several VHDL source files at one time for a large, complex design, use the `-f` option.

The syntax is

```
vhdlin -f <filename>
```

where

<code><filename></code>	is the name of the file containing the source file names.
-------------------------------	---

You must import the entity files before the architecture files.

For example, if you want to import four files at the same time, use the command

```
vhdlin -param param_file -f file
```

where the contents of *f_file* are

block_entity.vhd

block_arch.vhd

top_entity.vhd

top_arch.vhd

TDM and Imported VHDL Design Libraries

If you collaborate with a team in the development of VHDL design libraries, you can manage the process of updating these libraries with Team Design Manager (TDM). TDM is useful whether you manage your libraries for a single release or throughout multiple releases.

The Team Design Manager User Guide gives more detailed information about the process of managing libraries with TDM.

You can also use TDM to manage the libraries you import into VHDL In. When started, VHDL In first determines whether an import library is managed by TDM.

VHDL In operation in TDM Mode

VHDL In calls the ncvhdl parser to parse the VHDL source files. By default, VHDL In uses the target library as the parsing area.

In TDM operation, specify a work area other than VHDL In import library so that parsed intermediate data does not get checked in along with the design data created by VHDL In.

Checking a Library into TDM

To check a VHDL design library into TDM

- At the command prompt, enter

```
tdmcheckin -i 'libName'
```

The file is now managed by TDM for successive runs of VHDL In.

Enabling VHDL93 features

VHDL In calls the `ncvhdl` parser to parse the input VHDL files. When you want the parser to support the features of VHDL93, use the `-v93` option in the VHDL In command line or specify the following information in the `hdl.var` file:

```
DEFINE NCVHDOPTS -v93
```


Glossary

A

analysis

Compilation of the VHDL source code.

architecture

Describes the implementation of an entity. The architecture specifies to the compiler what the device really does, and how it achieves this. A single entity can have several different architectures, and each architecture must have an associated entity.

B

behavioral constructs

VHDL expressions that describe the behavior of a component.

binding specification

The assignment of specific components to specific entities. Configurations control binding.

C

CDF

Component Description Format. A system for dynamically changing and storing parameters and displaying information for components and sets of components for different versions (levels) of designs.

OA

Cadence OpenAccess database, the design data storage format used by Cadence tools.

cell

The Cadence software representation of a design element or component. A component of a design; a collection of different aspects (representations) of the component's implementations, such as its schematic, layout, or symbol representations. A design object consisting of a set of views that can be stored and referenced independently. A cell can include other cells, forming a hierarchical design. A cell is an individual building

block of a chip or system. In the database, a cell contains all the cellviews of that cell. An inverter and a buffer are examples of a small cell. A decoder register, ALU (arithmetic logic unit), memories, complete chips, and printed circuit boards are examples of large cells.

cellview

A specific representation (view) of a cell. A particular representation of a particular component, such as a D flip-flop's physical layout or a NAND gate's schematic symbol. A database object containing all the information unique to a particular representation of a particular component. See also [view](#).

CIW

Command Interpreter Window, the main Virtuoso® Design Environment window within the Cadence software.

.cod

Short for code (machine instruction streams).

Command Interpreter Window (CIW)

The initial window within the Cadence software. You can enter SKILL commands in the CIW command line. CIW is the main Virtuoso Studio Design Environment window within the Cadence software.

compilation

See [analysis](#).

component

A fundamental unit within a system that encapsulates behavior or structure (also known as an *element*). A cell, with cellviews and associated CDF.

configuration declaration

Selects which units are used in each level of the design hierarchy.

construct

A statement or basic element in a programming language.

context clause

A statement in a design unit that identifies which elements in which libraries the design unit references.

cyclic field

A button on a form that displays a list of valid options when you hold a mouse button down on top of it. For example, one cyclic field button enables you to select the units of measurement, and lists inches, centimeters, mils, or microns.

D

Virtuoso Studio Design Environment

The set of functions, commands, windows, menus, and databases that comprise the entire graphics environment, beginning with the CIW.

design unit

The basic building blocks of a VHDL design, including entity, architecture, package, package body, and configuration declarations.

destination library

The CDBA format library where the imported design is placed.

E

entity

The interface to a component. Entities communicate through generic devices and ports.

entity declaration

Describes the interface to a component. Entities communicate through generic devices and ports.

enumeration literal

A variable used for counting.

F

form

A window or dialog box enabling you to specify information and various options pertaining to a specific command or menu item. The options take effect when you click *OK* or *Apply*.

form field

The area on a form where you indicate values, names, and selections.

H

hierarchy

The nested design levels, such as instances within a cell. By default, you open the top level in the hierarchy when you open a cellview.

I

IEEE

Institute of Electrical and Electronic Engineers.

instance

A copy of a master symbol in a design.

IR

Intermediate representation.

L

LHS

Left-hand side.

library

A logical collection of cells, views, and technology information. A physical collection of files and directories that can reside anywhere in the file system. A library can be shared by all users and controlled by a single person.

Library Browser

The Cadence window displaying the list of available libraries in your search path as well as any open libraries. You can search through your libraries, cells, views, cellviews, and versions with the Library Browser.

list box field

Fields in forms that you can scroll up or down to view and select items from a list.

LSE

Language Sensitive Editor. A text editor with features specific to one programming or design language, such as syntax checking.

M

menu banner

The rectangular area across the top of a window. It contains menus, such as Open and Design Manager.

N

ncvhdl

Cadence's NC-VHDL parser.

netlist

The cells and interconnections that comprise the logical design of a circuit.

O

OSS

Open Simulation System.

P

package, package body declaration

Contain commonly used declarations such as constants, type declarations, and global subprograms for multiple design units.

parameter

A characteristic of a component.

PI

Procedural Interface.

primary design unit

Entity, package, and configuration declarations. Primary DUs (design units) must be analyzed (compiled) before secondary DUs.

R

resource library

A library that contains library units that are referenced in the design unit undergoing analysis. While there can be only one working library, there can be many resource libraries. Resource libraries are referenced by designs.

RHS

Right-hand side.

RISC

Reduced Instruction Set Computer.

RPU

RISC Processor Unit.

S

secondary design unit

Architecture and package body declarations. Secondary DUs (design units) must be analyzed (compiled) after primary DUs.

SKILL

A proprietary Cadence programming language based on over 1,200 functions. Each function follows a standard SKILL syntax and acts on a particular data type or data object.

SKILL function

The fundamental element in the SKILL language. An operation to be performed (usually specific to Cadence software), usually followed by the items or data objects that the operation is to be performed on.

SIR

Structural intermediate representation.

V

VHSIC

Very High Speed Integrated Circuits.

VHDL

VHSIC Hardware Description Language.

view

A specific representation of a cell, such as schematic, geometric, symbolic, logical, and routing. In the database, a view contains all the cellviews of that view. Each view can have a *viewType* property that associates it with a specific application. For example, the view named “XYZ” could be a *viewType* “layout.” See also [cellview](#).

viewType

A property of a view that associates it with a particular application. Virtuoso® Design Environment recognizes a set of registered viewTypes, such as schematic and layout views.

W

window

In the X environment, a rectangular area on a graphics workstation that emulates a terminal and runs an application separate from the applications in other windows. Usually you can have several windows on your screen at one time.

working library

The library where a library unit is placed after that library unit is created from the analysis of a design unit. There can be only one working library. The working library holds the design itself.