# Contents

**Preface**  ................................................................. 7  
**About This Manual**  .................................................. 7  
  - **Finding Information in This Manual**  ........................ 7  
  - **Conventions**  .................................................... 8  
**Related Documents**  .................................................. 8  
  - **Related Manuals**  ................................................. 8  
  - **Customer Support**  ............................................... 9  
**Syntax Conventions**  ................................................ 9  
  - **SKILL Syntax Example**  ........................................ 10  

1  
**Introducing VHDL In**  ................................................ 11  
**Uses for VHDL In**  .................................................... 11  
**Other Cadence Products Used with VHDL In**  ....................... 12  
**Design Flow Using VHDL In**  ....................................... 12  
**VHDL In Software Directory**  ...................................... 13  
**Starting Options and Requirements**  ................................ 13  
  - **Starting Options**  ................................................ 13  
  - **Input Requirements**  ............................................. 14  
**Conversion Limitations**  ............................................. 14  

2  
**Getting Started with VHDL In**  .................................... 15  
**Introduction**  ......................................................... 15  
**Setting Up the Library Environment**  ................................ 15  
  - **Setting Up the cds.lib File**  .................................. 15  
  - **Using the Text Editor**  ........................................ 16  
  - **Using the CIW**  .................................................. 17
Guide to VHDL In for Design Framework II

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting VHDL In in Standalone Mode</td>
<td>22</td>
</tr>
<tr>
<td>Starting VHDL In with Design Framework II</td>
<td>23</td>
</tr>
<tr>
<td>CIW Menu Options</td>
<td>23</td>
</tr>
<tr>
<td>SKILL Function</td>
<td>27</td>
</tr>
<tr>
<td><strong>3</strong></td>
<td></td>
</tr>
<tr>
<td>VHDL In Forms</td>
<td>29</td>
</tr>
<tr>
<td>VHDL Import Form</td>
<td>30</td>
</tr>
<tr>
<td>Import Options</td>
<td>31</td>
</tr>
<tr>
<td>Power</td>
<td>32</td>
</tr>
<tr>
<td>Ground</td>
<td>33</td>
</tr>
<tr>
<td>Schematic Generation Options Form</td>
<td>33</td>
</tr>
<tr>
<td><strong>4</strong></td>
<td></td>
</tr>
<tr>
<td>Importing a Simple VHDL Design</td>
<td>38</td>
</tr>
<tr>
<td>Introduction to the Example Design</td>
<td>38</td>
</tr>
<tr>
<td>Checklist Before Importing a Design</td>
<td>39</td>
</tr>
<tr>
<td>Setting Up the Library Environment</td>
<td>39</td>
</tr>
<tr>
<td>Importing to VHDL Cells</td>
<td>41</td>
</tr>
<tr>
<td>Setting Up the VHDL Import Form</td>
<td>42</td>
</tr>
<tr>
<td>Processing Generate Statements</td>
<td>47</td>
</tr>
<tr>
<td>Creating a New Target Library</td>
<td>48</td>
</tr>
<tr>
<td>Viewing the Results</td>
<td>49</td>
</tr>
<tr>
<td>Importing to a Netlist</td>
<td>52</td>
</tr>
<tr>
<td>Setting Up the VHDL Import Form</td>
<td>53</td>
</tr>
<tr>
<td>Creating a New Target Library</td>
<td>55</td>
</tr>
<tr>
<td>Viewing the Results</td>
<td>55</td>
</tr>
<tr>
<td>Importing to a Schematic</td>
<td>58</td>
</tr>
<tr>
<td>Setting Up the VHDL Import Form</td>
<td>58</td>
</tr>
<tr>
<td>Creating a New Target Library</td>
<td>61</td>
</tr>
<tr>
<td>Viewing the Results</td>
<td>61</td>
</tr>
</tbody>
</table>
5
Importing a Complex VHDL Design (RISC Processor Unit). 65

Overview ................................................................. 65
RPU Design Structure .................................................. 66
Setting Up the VHDL Import Form ............................... 66
Specifying Schematic Generation Options ...................... 70
Creating a Target Library ............................................ 72
Viewing Error and Log Files ....................................... 72
Viewing the Results ................................................... 73
  Accessing the Symbol Editor ...................................... 73
  Viewing the Schematic .............................................. 75

6
Conversion Issues ..................................................... 81
Introduction ............................................................ 81
Binding Issues ........................................................ 81
  One Component, Multiple Entity/Architecture Pairs .......... 82
  One Entity, Multiple Components ................................ 82
Case Sensitivity Issues .............................................. 83
  Possible VHDL Design Import Errors .......................... 84
  Object Search in CDBA ........................................... 84
Other Issues ........................................................... 87
  Signal Declarations ................................................ 87
  Vector Nets .......................................................... 87
  Noninteger Vector Indices ....................................... 87
  Buffer Ports .......................................................... 88
  Port Conversion Functions ...................................... 89
  Attribute Specifications ......................................... 89
  Importing Generics ............................................... 89
  Component Declarations ......................................... 91
  Port Maps ............................................................ 92
  Concurrent Assignment Statements ............................ 92
  Behavioral View for Continuous Signal Assignment Statements ........................................... 93
  Power and Ground Signals ........................................ 93
Guide to VHDL In for Design Framework II

Nontranslatable Structural VHDL Constructs .............................................. 93
  Port Subtypes and Modes ........................................................................ 94
  Constructs in Architectures ........................................................................ 96

7
Error Messages ............................................................................................... 98
Dialog Boxes .................................................................................................. 98
  File Name Does Not Exist ........................................................................... 98
  No Target Library Name Specified ............................................................... 99
  Library Name Not Found ............................................................................. 99
  Import Completed Successfully ................................................................... 100
  Import Completed with Errors/Warnings ..................................................... 100
  Import Aborted When Reached Maximum Errors ......................................... 100
Text String Error Messages ........................................................................... 101
  Overview ..................................................................................................... 101
  Line Command Errors ............................................................................... 101
  Analyzer Errors .......................................................................................... 102
  Parameter File Parsing ............................................................................... 102
  Composer Design Library ........................................................................... 103
  Memory ......................................................................................................... 103
  General Errors ............................................................................................. 103
Using the VHDL Tool Box Status Window .................................................. 103

8
Creating a Parameter File ................................................................. 106
Parameter File Structure .................................................................... 106
Description of Parameters .................................................................. 107

9
VHDL In Standalone Options ......................................................... 114
Introduction ......................................................................................... 114
Getting Help on a VHDL In Command ............................................... 115
Displaying the Version Number of VHDL In ...................................... 116
Suppressing Printing of the Copyright Banner ................................... 116
Hiding the Display of Schematic Extraction Errors ........................... 117
Specifying a cds.lib File ..................................................................... 117
Compiling Functional Cellviews ....................................................... 117
Specifying the VHDL WORK Library ................................................ 118
Ignore Extra Pins on Symbols ............................................................. 118
Speeding Up Run Time ....................................................................... 118
Specifying a Schematic Parameter File ............................................. 119
Specifying Multiple VHDL Source Files ......................................... 119
TDM and Imported VHDL Design Libraries ...................................... 120
  VHDL In operation in TDM Mode ................................................... 120
  Checking a Library into TDM ......................................................... 120
  Enabling VHDL93 features ............................................................ 121
  Enabling Vendor Compatibility ..................................................... 121

Glossary .................................................................................................. 122

Index ...................................................................................................... 138
Preface

This preface discusses the following:

- “About This Manual” on page 7
- “Related Documents” on page 8
- “Syntax Conventions” on page 9

About This Manual

To use VHDL® In For Design Framework II™User Guide, you should be familiar with designing and simulating electronic designs based on VHDL with Cadence design and simulation software.

Finding Information in This Manual

The following table summarizes the topics covered in this manual.

<table>
<thead>
<tr>
<th>If you want to . . .</th>
<th>Read . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find out what VHDL In is</td>
<td>Chapter 1, “Introducing VHDL In”</td>
</tr>
<tr>
<td>Start VHDL In</td>
<td>Chapter 2, “Getting Started with VHDL In”</td>
</tr>
<tr>
<td>Learn about VHDL In forms</td>
<td>Chapter 3, “VHDL In Forms”</td>
</tr>
<tr>
<td>Import a VHDL design</td>
<td>Chapter 4, “Importing a Simple VHDL Design”</td>
</tr>
<tr>
<td>Import a more complex design and generate a schematic</td>
<td>Chapter 5, “Importing a Complex VHDL Design (RISC Processor Unit)”</td>
</tr>
<tr>
<td>Know how VHDL In handles difficult conversion issues</td>
<td>Chapter 6, “Conversion Issues”</td>
</tr>
<tr>
<td>Find out about a particular error message</td>
<td>Chapter 7, “Error Messages”</td>
</tr>
<tr>
<td>Learn the rules for creating a parameter file</td>
<td>Chapter 8, “Creating a Parameter File”</td>
</tr>
</tbody>
</table>
Conventions

Boxes and arrows in a sequence like the one below show you the order in which you select a command from a menu.

```
File => Import ...
```

Related Documents

For more information about VHDL In and other related products, you can consult the sources listed below.

Related Manuals

You use VHDL In with other Cadence products during the design process. The following give you more information about these products.

- **Composer: Design Entry Help** describes the Composer design entry product.
- **Composer: VHDL Interface User Guide** describes how to use the rest of the VHDL Interface product, including the VHDL Tool Box, that supports using VHDL with Composer™.
- **Leapfrog VHDL Simulator Reference** describes the Leapfrog™ VHDL simulator.
- **Leapfrog VHDL Simulator User Guide** describes the process of simulating a sample design with Leapfrog.
- **Notebook User Guide** describes how to use the Notebook graphical user interface to start tools and run utilities, including VHDL In.
- **Team Design Manager User Guide** describes how to use the Team Design Manager to manage the process of updating VHDL design libraries.
Customer Support

There are several ways to obtain support when using Cadence products.

- Cadence offers many classes. Ask your sales representative for more information.
- The Cadence Customer Response Center (CRC) hotline telephone number operates from 8 a.m. to 8 p.m. Eastern Time:
  1-800-CADENC2 (1-800-223-3622)
- You can also send e-mail to the CRC:
  crc_customers@cadence.com

Syntax Conventions

This list describes the syntax conventions used for the SKILL™ functions specified in this manual.

\textbf{literal} \texttt{(LITERAL)} \hspace{1cm} Nonitalic (UPPERCASE) words indicate keywords that you must enter literally. These keywords represent command (function, routine) or option names.

\textbf{argument} \texttt{(z_argument)} \hspace{1cm} Words in italics indicate user-defined arguments for which you must substitute a name or a value. (The characters before the underscore (_\_\_\_) in the word indicate the data types that this argument can take. Names are case sensitive. Do not type the underscore (z_\_\_) before your arguments.)

\texttt{[ ]} \hspace{1cm} Brackets denote optional arguments.

\texttt{...} \hspace{1cm} Three dots (\ldots) indicate that you can repeat the previous argument. If you use them with brackets, you can specify zero or more arguments. If they are used without brackets, you must specify at least one argument, but you can specify more.

argument\ldots specify at least one, but more are possible

[argument]\ldots you can specify zero or more

,\ldots \hspace{1cm} A comma and three dots together indicate that if you specify more than one argument, you must separate those arguments by commas.
If a command line or SKILL expression is too long to fit inside the paragraph margins of this document, the remainder of the expression is put on the next line, indented. For example:

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

When writing the code, you should always put a backslash (\) at the end of any line that continues on to the next line.

**SKILL Syntax Example**

The following example shows typical syntax characters used in SKILL.

```vhdl
list( g_arg1 [g_arg2] ... ) => l_result
```

This example illustrates the following syntax characters:

- `list`  
  Plain type indicates words that you must enter literally.

- `g_arg1`  
  Words in italics indicate arguments for which you must substitute a name or a value.

- `()`  
  Parentheses separate names of functions from their arguments.

- `_`  
  An underscore separates an argument type (left) from an argument name (right).

- `[]`  
  Brackets indicate that the enclosed argument is optional.

- `=>`  
  A right arrow points to the return values of the function. Also used in code examples in SKILL manuals.

- `...`  
  Three dots indicate that the preceding item can appear any number of times.
Introducing VHDL In

This chapter discusses the following:

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

Uses for VHDL In

When you use VHDL In for Design Framework II™, you can convert a VHDL structural or behavioral description into one of three forms in Cadence database access (CDBA) storage format:

- Schematic view
- Netlist view
- VHDL text views.

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

You can import the following into a Design Framework II library:

- A VHDL design
- A VHDL ASIC library
- A VHDL design, minus modules that already exist in the Design Framework II library
- Pieces of a hierarchical design
A combination of the above

If you convert your VHDL design into schematics, you can edit the schematics with the Composer schematic editing tool. A separate product is available that converts VHDL structural architecture into schematics compatible with the Concept™ design entry product.

Other Cadence Products Used with VHDL In

VHDL In is available as part of the VHDL Interface product or by itself. VHDL Interface in Design Framework II is available as part of the VHDL Composer software package, or in combination with only the Schematic Generator.

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

VHDL In uses the Leapfrog Analyzer and the dfilbase software. All designs are analyzed by Leapfrog Analyzer before they are brought into VHDL In.

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 .ast (abstract syntax tree) file generated by Leapfrog Analyzer is an intermediate representation of a VHDL design. The .cdb file generated by VHDL In is in CDBA (Cadence database access) format, the design data storage format used by Cadence tools.
VHDL In Software Directory

The VHDL In software is located in

\(<your\_install\_dir>\)/tools/dfII/bin/vhdlin

where

\(<your\_install\_dir>\) is a variable signifying the directory where your Cadence software is installed.

To find out where the Cadence software is installed, enter \(\text{cds\_root}\) 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 \(\text{cds\_root}\):

\(\text{cds\_root\ executableName}\)

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:

\(\text{Error! Cant determine installation root from PATH}\)

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 menus in the Design Framework II Command Interpreter Window (CIW).

In the standalone mode Chapter 2, “Getting Started with VHDL In,” describes these options in more detail.

Input Requirements

Before starting VHDL In in Design Framework II, 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 Composer schematics or CDBA 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 6, “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 Design Framework II

Introduction

You can start VHDL In two ways:

■ From the Command Interpreter Window (CIW), the Cadence graphical user interface for Design Framework II
■ In standalone mode, at 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 Design Framework II 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 exactly as you see here.)

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

   DEFINE basic <your_install_dir>/tools/dfII/etc/cdslib/basic
   DEFINE sample <your_install_dir>/tools/dfII/samples/cdslib/sample
   DEFINE US_8ths <your_install_dir>/tools/dfII/etc/cdslib/sheets/US_8ths
   DEFINE std <your_install_dir>/tools/leapfrog/files/std
   DEFINE ieee <your_install_dir>/tools/leapfrog/files/IEEE

3. Add additional libraries you want to use.
Example entries in a `cds.lib` file for additional design libraries are

```bash
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 Design Framework II 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.

1. Enter in an xterm window

   `icds &`
The CIW appears.
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.

<table>
<thead>
<tr>
<th>Library</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>/u/4.4.1/tools/dfii/etc/cdslib/basic</td>
</tr>
<tr>
<td>samples</td>
<td>/u/4.4.1/tools/dfii/samples/cdslib/sample</td>
</tr>
<tr>
<td>US_8ths</td>
<td>/u/4.4.1/tools/dfii/etc/cdslib/sets/US_8ths</td>
</tr>
<tr>
<td>std</td>
<td>/net/cds9066/usrl/cds/4.4.1/4.4.1.27/tools/leapfrog/files/std</td>
</tr>
<tr>
<td>ieee</td>
<td>/net/cds9066/usrl/cds/4.4.1/4.4.1.33/tools/leapfrog/files/IEEE</td>
</tr>
<tr>
<td>vhdl_design</td>
<td>/net/cds10033/ul/klunde/vhdl_design</td>
</tr>
<tr>
<td>vhdlin_design</td>
<td>/net/cds10033/ul/klunde/vhdlin_examples/pspcodesign/vhdlin_design</td>
</tr>
<tr>
<td>rpu_lib</td>
<td>/net/cds10033/ul/klunde/vhdlin_examples/cpu_design/cpu_lib</td>
</tr>
<tr>
<td>sysdesign</td>
<td>/net/cds10033/ul/klunde/vhdlin_examples/sysdesign</td>
</tr>
</tbody>
</table>

2. 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 Leapfrog performs on the reference libraries when it analyzes a design you are importing.


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
   
   icds &
The CIW appears.

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

The New Library form opens. Enter the name of the new library and a path to its location.

Enter the library name.

Enter the path to the library.

Click on the *Don’t need a techfile* button.

You can use the scroll bar in the directory window to fill in the library path field. Click on a directory name to enter it into the path field.
2. In the Name text field, specify the library name.
   xyzdesign

3. In the field below the Directory list box, enter the path to the new library.
   
   \texttt{<your\_install\_dir>/xyzdesign}

   \texttt{<your\_install\_dir>} is a variable signifying the particular path you use for the Cadence installation directory.

4. Click \textit{Don't need a techfile}.

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

5. Click \textit{Apply}.

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

   When you create the last new library, click \textit{OK} instead of \textit{Apply}, to close the New Library form.

   The new library is created.

6. In the CIW, select \textit{Tools – Library Manager}.

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

7. Click on the name of the new library.

   The contents of the new library (\textit{ieee}) appear in the cell display window.
This example has *Show Categories* turned off, and shows the references to the five required libraries (*basic*, *sample*, *US_8ths*, *std*, and *ieee*) and to the one user-defined library *cdsDefTechLib*.

*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 Design Framework II running (no CIW is displayed), you can run VHDL In in standalone mode.

➤ In a terminal window, type in the following command:

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

where

the `-pfns` argument to the `-param` function 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 \(-f\) function 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.

\section*{Caution}

\textit{If you have an NC set up for ToolBox, and want to run vhdlin from the commandline, you need to ensure that while running import, your cds.lib file should not have any reference to NC libraries. It should only refer to Leapfrog libraries. To revert back to the NC set up for ToolBox, you should remove the references to Leapfrog libraries and include references to the NC libraries.}

\section*{Caution}

\textit{All VHDL users who use the ToolBox UI are transparent to these changes. The ToolBox will take care of the cds.lib file.}

\section*{Starting VHDL In with Design Framework II}

To start VHDL In with Design Framework II

\begin{itemize}
  \item Enter in an xterm window
    \begin{verbatim}
    icds &
    \end{verbatim}
  \end{itemize}

The Command Interpreter Window (CIW) for Design Framework II 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 \texttt{SKILL} command at the command prompt in the CIW.

\section*{CIW Menu Options}

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

\begin{itemize}
  \item \textit{File – Import – VHDL}
  \item \textit{Tools – VHDL Toolbox}
\end{itemize}
File-Import-VHDL

One method of accessing VHDL In is through the CIW.

Go to the File menu on the CIW and select **File – Import – VHDL**.

Select this option.
The VHDL Import form opens.

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

The other method of accessing the VHDL Import form through the CIW is with the VHDL Tool Box form.

Go to the Tools menu in the CIW and select Tools – VHDL Tool Box.

Select this option.
The VHDL Tool Box form appears.

1. Go to the Commands menu and select *Commands – Import*.

The **VHDL Import form** opens.

**SKILL Function**

`vhdlHiImport()`
Enter this SKILL function at the CIW command prompt to open the VHDL Import form.
VHDL In Forms

This chapter discusses the following:

- VHDL Import Form
- Schematic Generation Options Form
**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 Import Form](image)
The parameters corresponding to many options and fields in this form are described in Chapter 8, “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.

*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.

**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 Composer schematic that graphically represents the VHDL architecture.
- *netlist* generates a CDBA 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 option is selected by default. The vhdl view is always overwritten.

This corresponds to the overwriteExistingView parameter in a parameter file.

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 compiled by the Leapfrog compiler cv. The option is deselected by default.

Compiler Options is the field where you specify the Leapfrog compiler 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 Leapfrog Analyzer uses to store the intermediate results of the import process (the .ast files). 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.

**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 Form

Use the Schematic Generation Options form to generate a schematic view of your design. You access this form from the VHDL Import form. The parameters corresponding to many options and fields in this form are described in Chapter 8, “Creating a Parameter File.”
On the VHDL Import form, click on the Schematic Generation Options bar at the bottom of the form.

Click on this bar
The Schematic Generation Options form appears.

Sheet Border Size 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 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** 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** 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** 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** 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** 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** 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.
Full Place and Route 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 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 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 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 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.
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. 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 run VHDL In on the design three times, producing a different type of output each time.

You 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 Leapfrog analyze your design?

5. Have you modified the design to eliminate or change those constructs that VHDL In cannot convert? (See Chapter 6, “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 2, “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

   ~/vhdl_in_examples/simple_design/

   The content for each file follows.

**full_adder.a.vhd**

architecture structure of full_adder is

-- signal declarations
   signal temp_sum : bit;
   signal temp_carry_1 : bit;
   signal temp_carry_2 : bit;

-- local component declarations
   component half_adder
     port(x : in bit; y : in bit; sum : out bit;
         carry : out bit);
     end component;
   component or_gate
     port(in1 : in bit; in2 : in bit; out1 : out bit);
     end component;

begin

-- component instantiation statements
   U0 : half_adder
      port map(a, b, temp_sum, temp_carry_1);
   U1 : half_adder
      port map(temp_sum, carry_in, ab, temp_carry_2);
   U2 : or_gate
      port map(temp_carry_1, temp_carry_2, carry_out);

end structure;

**full_adder.e.vhd**

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;
Importing a Simple VHDL Design

half_adder.a.vhd
architecture behavior of half_adder is
begin
  process begin
    sum <= x xor y after 5 Ns;
    carry <= x and y after 5 Ns;
    wait on x,y;
    end process;
  end behavior;
end behavior;

half_adder.e.vhd
entity half_adder is
  port(
    x : in bit;
    y : in bit;
    sum : out bit;
    carry : out bit
  );
end half_adder;

or_gate.a.vhd
architecture behavior of or_gate is
begin
  process begin
    out1 <= in1 or in2 after 5 Ns;
    wait on in1,in2;
    end process;
  end behavior;
end behavior;

or_gate.e.vhd
entity or_gate is
  port(
    in1 : in bit;
    in2 : in bit;
    out1 : out bit
  );
end or_gate;

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. In the CIW, select *File – Import – VHDL*.
The VHDL Import form opens.

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

Find the design directory or file by clicking in the Files List Box.

Enter the target library name.

Use Add >> to move the design files to the target library display.

Set this to vhdl.

Complete the list of reference libraries.

//hn/padm/*.*.vhd

Import Options

Import Structural Architectures

Reference Libraries

Symbol View Name

Overwrite Existing Views

Case Sensitive Symbol Matching

Maximum Number of Errors

Compile VHDL Views After Import

Compiler Options

VHDL WORK Library Name

Summary File

Power

Net Name vddl Value '1'

Data Type std_ulogic

Ground

Net Name gnd Value '0'

Data Type std_ulogic
You can move down a directory either of the following ways:

- Enter a path to the directory in the *File Name* field and press *Return*.
- Double-click on 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 on ../ 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.

2. 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.

3. 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 on 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 adjacent) and add them simultaneously to the Import Files List Box. Drag the mouse across the file names to highlight them, and click *Add>>*.

4. 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
existing target library. A dialog box with an error message appears when you press Return without specifying a target library.

5. 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 Leapfrog, use only lowercase letters to name your target libraries.

6. Set the Import Structural Architectures As cyclic field to vhdl.

You can leave all other options at their default setting.

7. In the Reference Libraries field, add sample, std, and ieee to the list.
Your form should look like the sample form.

![Sample Form Image]

8. 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.

**Processing Generate Statements**

VHDL In can create a schematic view for a design unit containing statements with a `FOR` or `IF` generation scheme with locally static expressions. It can also create appropriate bus nets for signal declaration of multidimensional array of scalar objects.

**Example:** Simple `for generate` clause with locally static expression. Instantites component inside the `generate` block.

```vhdl
library datapathlib;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE datapathlib.new_types.all;
ENTITY gen_syn IS
  PORT(a1 : IN std_logic_vector(3 DOWNTO 0);
       a2 : IN std_logic_vector(3 DOWNTO 0);
       z : OUT std_logic_vector(3 DOWNTO 0));
END gen_syn;

ARCHITECTURE dpgen OF gen_syn IS
BEGIN
  inst1: for bit_index in 3 downto 0 generate
    cell_and: \AN2D1\;
    port map (\A1\=>a1(bit_index), \A2\=>a2(bit_index), \Z\=>z(bit_index));
  end generate inst1;
end dpgen;
```
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.
1. 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 *Don’t need a techfile*, 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. 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 if you want to view the error message file.

If you click Yes to either prompt, the VHDL Tool Box status window appears, displaying the error or log file.

In addition to viewing the log file for a successful import process, you have the option of viewing the resulting cellviews of your design.
Viewing the Error File

The following error file was generated for this design example.

If you receive an error message like the one displayed, go back to the VHDL Import form and try importing a different combination of files. In this example, leaving out the OR gate design file lets the import process finish successfully. The order in which you specify files to import does not have an impact on the import process.

For examples of error messages, see Chapter 7, “Error Messages.”

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

Viewing the Log File

When the import process has finished successfully, you can examine the log file to check the results of importing to VHDL. The log 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 log file also mentions the files that VHDL failed to import, and the reason for the failure. The following log file was generated for a successful VHDL to VHDL import process. You can view the log 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.

![VHDL Tool Box Log File](image)

To close the VHDL Tool Box status window

➤ Select File – Close.

The contents of the log 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.

**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 on 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 on 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, click on the entity cellview file name.

   A text editor window opens, displaying the contents of the entity cellview: a text file and directory for the entity file for the full_adder design.
The Language Sensitive Editor, or LSE (*emacs*) is the example text editor shown here (displaying the entity cellview for a half_adder design).

If you are using LSE and the text editor window is empty, use the File – Open command to open `~/vhdl_designs/full_adder/entity`.

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

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

The top half of the LSE text editor window displays the contents of the `vhdl.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 CDBA format netlist. If the VHDL Import form is still open from the first example, 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 on entries until the design directory is displayed.

   To move the display up one directory, click on ../. To display the contents of a directory, click on 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 Leapfrog, 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.

Enter the target library name.

Use *Add>>* to move design files to the target library display.

Set to netlist.
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 *Don’t need a techfile*, and click OK.

![New Library Form]

A message appears in the CIW display area:

Created library "netlist_library" as ".\vhdl_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. 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.

If you click Yes to either prompt, the VHDL Tool Box status window appears, displaying the error or log file.

In addition to viewing the log file for a successful import process, you have the option of viewing the resulting cellviews of your design.
Viewing the Error 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 7, “Error Messages.”

To close the VHDL Tool Box status window

➤ Select *File – Close*.

Viewing the Log File
When the import process has finished successfully, you can examine the log file to check the results of importing to a netlist. The log 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 log file also mentions the files that VHDL fails to import, and the reason for the failure.

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

To close the VHDL Tool Box status window
Select File – Close.

The contents of the log 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.

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

4. Click the structure cellview name to display the structure directory containing the netlist files.
VHDL In places the VHDL design netlist in the `netlist.cdb`. These files are in the CDBA format required by Cadence tools. The following is the structure directory under the `full_adder component`.

```
drwxrwxr-x 2 user 512 Jun 30 16:16 .
drwxrwxr-x 5 user 512 Jun 30 16:16 ..
-rw-rw-r-- 1 user 40 Jun 30 16:16 master.tag
-rw-rw-rw- 1 user 1483 Jun 30 16:16 netlist.cd%
-rw-rw-rw- 1 user 1525 Jun 30 16:16 netlist.cdb
-rw-rw-r-- 1 user 86 Jun 30 16:16 pc.db
-rw-rw-rw- 1 user 532 Jun 30 16:16 prop.xx
-rw-rw-r-- 1 user 2968 Jun 30 16:16 vhdl.ast.a.203
lrwxrwxrwx 1 user 54 Jun 30 16:16 vhdl.vhd -> ../../..../../../usr1/vhdlin/designs/full_adder.a.vhd
```

**Importing to a Schematic**

In this section, you import the full adder design as a Composer schematic. Start at step 3 if you left the VHDL Import form open from the previous example.

**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 on ../. To display the contents of a directory, click on 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 Leapfrog, 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 Schematic Generation Options form. 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 on 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 Don’t need a techfile, and click OK.

After the new library is created, go back to the VHDL Import form and 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. 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.

If you click Yes to either prompt, the VHDL Tool Box status window appears, displaying the error or log file.

In addition to viewing the log file for a successful import process, you have the option of viewing the resulting cellviews of your design.

Viewing the Error 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 7, “Error Messages.”
Guide to VHDL In for Design Framework II
Importing a Simple VHDL Design

To close the VHDL Tool Box status window

➤ Select File – Close.

Viewing the Log File

When the import process has finished successfully, you can examine the log file to check the results of importing to a schematic. The log 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 log file also mentions the files that VHDL failed to import, and the reason for the failure.

You can view the log 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 log file looks similar to this:

To close the VHDL Tool Box status window

➤ Select File – Close.

The contents of the log 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.
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, click the structure cellview file name.
The Composer schematic editor opens, displaying the contents of the *structure* cellview: a schematic for the *full_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 Composer Symbol Editor window.

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

The Composer Symbol Editor 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 Schematic Generation Options form 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 2, “Getting Started with VHDL In.”
RPU Design Structure

This sample design is included in the Cadence software hierarchy, at 
<your_install_dir>/tools/dfII/samples/vhdlin, where 
<your_install_dir> is your 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 the VHDL Tool Box, this example starts from the CIW. The example design is located in /usr1/vhdlin/example. Before you start, copy the
1. In the CIW, select File – Import – VHDL

Design example from the `<your_install_dir>/tools/dfII/samples/vhdlin` directory and keep it in your own account.

```plaintext
1. In the CIW, select File – Import – VHDL
```
The VHDL Import form opens.

2. Double-click on entries in the Files List Box until the design directory is displayed.
   
   To move the display up one directory, click on ../. To display the contents of a directory, click on 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 on 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.

  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.

You can distribute design units across multiple directories.

The directories can include a variety of .vhd files, but do not import non-VHDL files.

Repeating design unit file names does not prevent importation.
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 on the Schematic Generation Options bar.
   
   The Schematic Generation Options form opens.

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 easily on a C-size schematic.

6. Deselect Generate Square Schematics.
The standard Cadence schematic sheet frame is rectangular, so turning off this option improves the use of schematic sheet space.

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 Don't need a techfile.
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"

1. Click in the open VHDL Import form to activate it.
2. Click Apply again.

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.

The log includes a set of processing statements for each design element that is imported.

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, select Tools – Library Manager.
The Library Manager window opens.

1. Click on rpulib or enter rpulib in the Library text field.
2. Click on alu to display the cellviews.
3. To display the alu symbol, click on symbol.

2. In the Library column of the Library Manager window, enter rpulib in the Library text field and press Return or click on 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 on the alu cell name.

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

4. Double-click on the view name symbol.
The Symbol Editor opens, displaying the symbol for *alu*.

![Symbol Editor](image)

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

5. To close the Symbol Editor, select *Window – 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, click the *schematic* cellview of the *rputop* cell to view the schematic of the cell.
The Composer Schematic Editor opens, displaying the schematic of *rputop*.

VHDL In, following the criteria set up in the Schematic Generation Options form, 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 4.

Schematic sheet 4 (shown in this step) displays *rpustruc* and some I/O pins.
1. In the schematic window, select Design – Hierarchy – Descend Edit, and click on rpustruc.
   The Descend form opens.

2. In the index schematic, select Sheet 6 and click OK.
   The Composer Schematic Editor 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 places the design name in the title box of the schematic frame.
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.

1. To exit the design, select Window – 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 CDBA 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 actual 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.

```vhdl
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.

```vhdl
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 CDBA format are case sensitive, whereas the VHDL
library structure is case insensitive. The intermediate (.ast) files generated by Leapfrog do
not preserve the case of the original declaration in CDBA 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 the 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 CDBA 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.

  ```vhdl
  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 CDBA

Object search in CDBA 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 CDBA 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 CDBA 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 CDBA 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(...
II: 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:

```vhdl
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

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

and later use the component instantiation statement

```vhdl
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 CDBA

Object creation in CDBA preserves the original case of the declaration.
For example, the code

```vhdl
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 Composer database (CDBA).

**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 Composer 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 `bitToMvl` for the port map `(bitToMvl (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 `bitToMvl` for the port map `(A => bitToMvl(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 the 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 CDBA.

The following binding specification is imported as a schematic.

```vhdl
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.

```vhdl
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
\[
\text{selec}x \leftarrow \text{Sel}(2) \& \text{Sel}(1) \& \text{Sel}(0);
\]
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,
\[
a \leftarrow '1'
\]
indicates that net \(a\) is a power net, and
\[
b \leftarrow '0'
\]
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 CDBA 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 CDBA allows three port modes: *in*, *out*, and *inout*.

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

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 CDBA 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 CDBA port.

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

```vhdl
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 scalar type. CDBA ports can be either scalar or constrained single dimensional arrays of scalars. This port has no direct mapping to any CDBA object. Therefore, VHDL In cannot convert this port to a CDBA symbol.

Port b
The port is a record 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 CDBA symbol.

Port c
The port is a single-dimensional array of scalar type. The base type of the array is an unconstrained type. This port declaration creates an anonymous type. VHDL In can create this port in CDBA. Although the declarations for ports f and c are vector of size 10, the declaration for port c generates an anonymous type implicitly while the port f 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 CDBA has no parameterized ports, VHDL In cannot convert this port into a CDBA symbol.

Port e
The port is an array of a composite type (arrays and records). Different scalar elements of an array of composite type can be of different types. VHDL In cannot map such a port to a CDBA port.

Port f
The port is of mode linkage, which cannot be mapped to CDBA ports.
Port $g$ The port is a scalar type. This declaration implicitly creates an anonymous type. VHDL In can convert this port to a CDBA 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.

Actual 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. CDBA 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 CDBA represent mapping to blocks. Because CDBA 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 CDBA 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 dialog box opens.
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 dialog box opens.

![No target library name specified](image)

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.

![Library "bad_target_lib" was not found](image)
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.

![VHDL Import completed successfully dialog box]

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.

![VHDL Import completed with errors/warnings dialog box]

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.

![Image of error message]

**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:

```
vhdl: *<severityCode>,<errorNumber> <errMsgString>
    [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.
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.
Composer Design Library

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

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.
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.

![Search Form]

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

![Save As Form]

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

![Print Form]

**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. 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 Composer schematic.

```
logFileName := /tmp/vhdlImLob14298
errorFileName := /tmp/vhdlinErb14298
importLibraryName := adder
inputPinSymbol := basic ipin symbol
outputPinSymbol := basic oinp 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
```
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.

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

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

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

**importLibraryName.** Specifies the logical library in CDBA 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.

```vhdl
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.

```vhdl
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.

```vhdl
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.

**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 CDBA 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 CDBA. 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 on the Schematic Generation Options form.

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 on the Schematic Generation Options form.

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 on the Schematic Generation Options form.

fontHeight. A real number that specifies the height of wire labels in user units. You must use Composer 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 on the Schematic Generation Options form.
**lineLineSpacing**. A real number, representing the space, in inches, between two adjacent nets. The range is 0.125 to 0.625 inches.

```vhdl
lineLineSpacing := 0.2
```

Corresponds to the *Line to Line Spacing* field on the Schematic Generation Options form.

**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.

```vhdl
lineComponentSpacing := 0.125
```

Corresponds to the *Line to Component Spacing* field on the Schematic Generation Options form.

**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.

```vhdl
componentDensity := 0
```

Corresponds to the *Component Density* sliding field on the Schematic Generation Options form.

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

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

```vhdl
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:

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

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

```vhdl
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 on the Schematic Generation Options form. 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 a very large design, this gain is significant. The default is TRUE.

fullPlaceRouteSchematic := TRUE

Corresponds to the Full Place and Route click box on the Schematic Generation Options form.

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 on the Schematic Generation Options form.

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 on the Schematic Generation Options form.

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 on the Schematic Generation Options form.

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 6, “Conversion Issues.”

**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.

```vhdl
work_file := my_work_lib
```

Corresponds to the *VHDL Work Library Name* field on the Schematic Generation Options form.
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
- Suppressing Printing of the Copyright Banner
- 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

vhdl_in Specifies the command to start VHDL In in the standalone mode.

-param Indicates that a parameter file will be read.

param_file Specifies the schematic parameter file.

<option> Specifies name of the option you want to use.

<VHDL source files> Specifies the VHDL source files you want to import, using the options you specified.

Caution

If you have an NC set up for ToolBox, and want to run vhdl_in from the command line, you need to ensure that while running import, your cds.lib file should not have any reference to NC libraries. It should only refer to Leapfrog libraries. To revert back to the NC set up for ToolBox, you should remove the references to Leapfrog libraries and include references to the NC libraries.

Caution

All VHDL users who use the ToolBox UI are transparent to these changes. The ToolBox will take care of the cds.lib file.

Getting Help on a VHDL In Command

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

The syntax is

vhdl_in -help <VHDL In command>

where

vhdl_in 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:
Guide to VHDL In for Design Framework II
VHDL In Standalone Options

Usage: vhdlin -param <paramFileName> -f <fileName> <vhdlSourceFileNames>
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

-version Specifies the version of VHDL In available to you.

The output of this command is similar to the following:
@(#)CDS: vhdlin version 4.4.1 11/22/96 09:44 (darbari)$

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 +noxtrsch option at the command prompt.

The syntax is

```
vhdl -param param_file +noxtrsch 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 4, “Importing a Simple VHDL Design”), use this command:

```
vhdl -param_file -+noxtrsch 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

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

An example use of this option is

```
vhdl -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

```
vhdl -compile
```

An example use of this command is

```
vhdl -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 entity file for the block design.
```

Guide to VHDL In for Design Framework II
VHDL In Standalone Options

block_arch.vhd  Specifies the architecture 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

param_file  It is the schematic parameter file.

An example use of this option is
    vhdlin -param param_file top_entity.vhd top_arch.vhd

where

top_entity.vhd  It is the entity file for the top design.
top_arch.vhd    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

<filename>  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 `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 leapfrog analyzer to analyze the VHDL source files. By default, VHDL In uses the target library as the analysis area.

In TDM operation, specify a work area other than VHDL In import library so that analyzed 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 Leapfrog VHDL analyzer to parse the input VHDL files. When you want the analyzer 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 DULUTHOPTS -v93
```

or,

```
DEFINE CVOPTS -v93
```

**Enabling Vendor Compatibility**

VHDL In calls the Leapfrog VHDL analyzer to parse the input VHDL files. When you want the analyzer to enable the vendor compatability mode, use the -compatibility option in the VHDL In command line or specify the following information in the hdl.var file:

```
DEFINE DULUTHOPTS -comp
```

or,

```
DEFINE CVOPTS -comp
```
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 actually does, and how it achieves this. A single entity can have several
    different architectures, and each architecture must have an associated entity.

.ast, AST
    Abstract syntax tree. An intermediate representation of a design produced by a VHDL
    analyzer.

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.

CDBA
    Cadence database access, 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 Design Framework II 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.

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

Design Framework II
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 on 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

Leapfrog
Cadence’s VHDL analyzer and simulator.

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
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

VHISIC
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 its 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. Design Framework II recognizes a set of registered viewTypeTypes, 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.
## Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>+noxtrsch option</td>
<td>117</td>
</tr>
<tr>
<td>, . . in syntax</td>
<td>9</td>
</tr>
<tr>
<td>. . . in syntax</td>
<td>9</td>
</tr>
<tr>
<td>.ast files</td>
<td>12</td>
</tr>
<tr>
<td>.ast files</td>
<td>12</td>
</tr>
<tr>
<td>case sensitivity</td>
<td>83</td>
</tr>
<tr>
<td>defined</td>
<td>122</td>
</tr>
<tr>
<td>.cdb files</td>
<td>12</td>
</tr>
<tr>
<td>.vhd files</td>
<td>69</td>
</tr>
</tbody>
</table>

### A

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract syntax tree file</td>
<td>12</td>
</tr>
<tr>
<td>actual port</td>
<td>89</td>
</tr>
<tr>
<td>adder design</td>
<td>38</td>
</tr>
<tr>
<td>schematic</td>
<td>64</td>
</tr>
<tr>
<td>adding VHDL design libraries</td>
<td>16, 19</td>
</tr>
<tr>
<td>adding VHDL source files to import list</td>
<td>31</td>
</tr>
<tr>
<td>alu design, symbol cellview</td>
<td>75</td>
</tr>
<tr>
<td>analyzer errors</td>
<td>102</td>
</tr>
<tr>
<td>architecture, defined</td>
<td>122</td>
</tr>
<tr>
<td>arrays</td>
<td>94</td>
</tr>
<tr>
<td>globally static</td>
<td>94</td>
</tr>
<tr>
<td>multidimensional</td>
<td>94</td>
</tr>
<tr>
<td>of composite objects</td>
<td>94</td>
</tr>
<tr>
<td>unconstrained</td>
<td>94</td>
</tr>
<tr>
<td>AST. See .ast files</td>
<td></td>
</tr>
<tr>
<td>attribute specifications</td>
<td>89</td>
</tr>
</tbody>
</table>

### B

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>behavioral constructs, defined</td>
<td>122</td>
</tr>
<tr>
<td>binding</td>
<td>81</td>
</tr>
<tr>
<td>multiple components to one entity</td>
<td>82</td>
</tr>
<tr>
<td>one component to multiple entity/architecture pairs</td>
<td>82</td>
</tr>
<tr>
<td>specifications</td>
<td>96</td>
</tr>
<tr>
<td>defined</td>
<td>122</td>
</tr>
<tr>
<td>port maps on</td>
<td>92</td>
</tr>
<tr>
<td>blocks</td>
<td>97</td>
</tr>
<tr>
<td>buffer ports</td>
<td>88</td>
</tr>
</tbody>
</table>

### C

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C model files</td>
<td>69</td>
</tr>
</tbody>
</table>
Guide to VHDL In for Design Framework II

structure 51, 57, 58, 63, 76
symbol 51, 57, 63, 76
cellview types
  functional 52
  netlist 58
  schematic 63, 75
  symbol 75
cellviews
  defined 123
  functional 117
  compiling 117
  symbol 73
checking a library into TDM 120
CIW 14, 15
  starting VHDL In from 15
Close command 105
closing Errors/Warnings list window 105
Command Interpreter Window. See CIW commands, Composer schematic editor
  Descend Edit 78
  Go To Sheets 77
commands, VHDL Tool Box status window
  Close 105
  Log File 105
  Next Error File 104
  Previous Error File 104
  Print 105
  Save As 105
  Search 105
compilation, defined 123
-compile option 117
Compile VHDL Views After Import box 32
Compiler Options field 32
compiling
  functional cellviews 117
  imported VHDL files 32
  component declarations 91
Component Density field 36, 70
Component Description Format. See CDF component symbols
searching 85
with binding
  case-sensitive search 85
with no binding
  case-sensitive search 85
  case-sensitive search of repeated instantiations 85
  case-sensitive search restrictions 85
componentDensity parameter 111
components
binding of 81
  symbol view for 81
Composer design library errors 103
Composer schematic editor 12, 64, 77
Composer schematics 14
Composer tool 12
Concept design entry tool 12
concurrent assignment statements 92, 96
configuration declarations, defined 123
construct, defined 123
contAssignSymbol parameter 108
context clause, defined 124
copyright banner, hiding display of 116
creating a parameter file 106
creating cds.lib file
  automatically 16
  using CIW 17
  using text editor 16
creating design libraries 16
customer support information 9
cyclic field, defined 124

D

Descend Edit command 78
design
  analysis 12, 81
  compilation 12
  name 79
Design Framework II 14
defined 124
design libraries
  adding 16
  case sensitivity 84
  creating 16
  errors in recognizing 84
design units 39
defined 124
destination library. See target library dfllbase software 12
dialog boxes
import libraries. See target libraries
import process
  errors in 84
  required design unit declarations for 39
  viewing results of 49, 55, 61, 73
Import Structural Architectures As field
importing
  multiple VHDL source files
    sequentially 118
    simultaneously 119
  to a schematic 58
  to CDBA netlists 52
  to VHDL design cells 41
importLibraryName parameter 107
index sheet 77
inputPinSymbol parameter 107
instance, defined 125
instantiations, searching 85
integration request. See IR
IR, defined 125
italics in syntax 9

K
keywords 9

L
label placement 80
Language Sensitive Editor. See LSE
Leapfrog
  analyzer 12
    case sensitivity and 83
    specifying options 32
  compiler cv 32
  workarea, specifying 32
LHS, defined 125
libraries
  defined 125
  design
    creating 16
    reference 16
Library Browser, defined 126
library environment for VHDL In
  contents 15
  setting up 39
Library Manager
  tool 16, 17, 74
  window 22, 51, 63
Library Path Editor form
  display 18
  uses for 17
  library units
    logical library 84
    referenced 39
line command errors 101
Line To Component Spacing field 36
Line To Line Spacing field 36
lineComponentSpacing parameter 111
lineLineSpacing parameter 111
list box field, defined 126
literal characters 9
Log File command 105
log files 51, 62, 73
  command menu 104
  viewing 72
logFileName parameter 107
LSE
  defined 126
  displayed 52

M
maxError parameter 109
maximum number of columns
  described 36
  for large design 70
Maximum Number of Columns field 36, 70
Maximum Number of Errors field 32
maximum number of rows
  described 35
  for large design 70
Maximum Number of Rows field 35, 70
maxNoCols parameter 110
maxNoRows parameter 110
memory errors 103
menu banner, defined 126
menu commands, selection conventions 8
Minimize Crossovers field 37
minimizeCrossovers parameter 112
multiple sheet schematics 97

N
net routing 80
netlist.cdb file 58
netlists
  CDBA format 52
defined 126
importing to 52
New Library form 19
display 20
new target library 48, 55
Next Error File command 104
-nocopyright option 116
noninteger vector indices 87
nontranslatable structural VHDL
constructs 93

O

object creation in CDBA 86
object search in CDBA 84
Optimize Wire Label Location field 37
optimizeLabels parameter 112
or_gate cell 64
OSS, defined 126
output formats 38
outputPinSymbol parameter 107, 108
Overwrite Existing Views field 32
overwriteExistingView parameter 109

P

package body declaration, defined 126
package declaration, defined 126
packages 81, 83
referred 91
-param option 119
parameter file 39
creating 106
parsing errors 102
structure 106
parameters
componentDensity 111
contAssignSymbol 108
defined 126
errorFileName 107
fontHeight 110
fullPlaceRouteSchematic 112
groundType 109, 110
ignoreContassFunc 108
importLibraryName 107
inputPinSymbol 107
lineComponentSpacing 111
lineLineSpacing 111
logFileName 107
maxError 109
maxNoCols 110
maxNoRows 110
minimizeCrossovers 112
optimizeLabels 112
outputPinSymbol 107, 108
overwriteExistingView 109
pinPlacement 111
powerLiterals 109
powerNetName 109
powerType 109
referenceLibraries 108
sheetBorderSize 110
squareSchematics 112
structuralViewType 108
symbolViewName 108, 109
writeInRefLibs 108
patch
expression 92
symbol 92
PI, defined 126
Pin Placement field 36
pinPlacement parameter 111
port conversion functions 89
port maps 92
port modes
buffer 94
inout 94
linkage 94
nontranslatable 94
VHDL 94
port subtypes, nontranslatable 94
ports
case sensitivity 84
data type for 33
ame of 32
errors in recognizing 84
value of 32
power options 32
power signals. See power nets
powerLiterals parameter 109
powerNetName parameter 109
powerType parameter 109
Previous Error File command 104
primary design unit 127
Print command 105
Print form 105
problematic ports 94
properties
vhdlGenericDefList  89
vhdlActualPortFuncName  89
vhdlArchitectureName  82
vhdlAssignInstance  92
vhdlAttributeDefList  89
vhdlDataType  87
vhdlFormalPortFunc  89
vhdlLibraryNames  83
vhdlPackageComponents  92
vhdlPackageNames  83
vhdlPortType  88
vhdlVecIndexType  88

schematic generation options  59, 77
Schematic Generation Options form 33, 70
accessing 34
Component Density field 36
display 35
Font Height field 36
Full Place and Route field 37
Generate Square Schematics field 37
Line to Component Spacing field 36
Line to Line Spacing field 36
Maximum Number of Columns field 36
Maximum Number of Rows field 35
Minimize Crossovers field 37
Optimize Wire Label Locations field 37
Pin Placement field 36
Sheet Border Size field 35

record types  94
reference libraries  81
adding
using Library Path Editor form 17
using text editor 16
case-sensitive search 86
creating 16
for symbols, specifying 31
Reference Libraries field 31
referenced library units 39
referenceLibraries parameter 108
referred packages 91
removing VHDL source files from import list 31
resource libraries 83
defined 127
RISC processor unit (RPU). See RPU
RISC, defined 127
RPU
defined 127
design structure 66
importing 65
runtime, speeding up 118

S
Save As command 105
Save As form 105
saving error or log file 105
schematic editing 12
schematic extraction errors, hiding display of 117
schematics
changing display of 65
component density 36
font height 36
full place and route 37
importing to 58
line-to-component spacing 36
line-to-line spacing 36
maximum number of columns 70
maximum number of rows 70
minimizing net crossovers 37
multiple sheet 97
number of columns 36
number of rows 35
optimizing wire locations 37
pin placement 36
sheet border size 35
sheet space for 70
square 37
viewing 75
Search command 105
Search form 105
searching
component reference libraries 86
component symbol instantiations with no binding 85
component symbols
Guide to VHDL In for Design Framework II

with binding 85
with no binding 85
with no binding, restrictions 85
in error or log file 105
reference libraries in all uppercase 86
selecting VHDL source files for import 31
setting up cds.lib file
  using Library Path Editor form 17
  using text editor 16
sheet border size 35
Sheet Border Size field 35, 70
sheet border size for large design 70
sheet space for large design 70
sheetBorderSize parameter 110
signal declarations 87, 96
SIR, defined 127
SKILL
  defined 127
  functions 28
    syntax conventions 10
specifications
  binding 96
  disconnection 96
specifying output format for imported files 31
specifying schematic generation options 70
-speedUp option 118
squareSchematics parameter 112
standalone mode 39
starting VHDL In 13
  in Design Framework II requirements 14
  in standalone mode 22
  with Design Framework II 23
structuralViewType parameter 108
structure cellview for full_adder design 63
Symbol Editor
  accessing 73
  window 64
symbol view 64
  for alu 74
  of a cell 73
Symbol View Name field 32
symbols, generating 32
symbolViewName parameter 108, 109
Synergy 12
syntax conventions 9

T
target libraries
  analysis in 117
  creating 48, 55, 61, 72
  defined 124
  managed by TDM 117
  not managed by TDM 117
  overwriting existing views 32
  specifying names for 31
Target Library name field 31
TDM
  checking VHDL library into 120
  workarea
    specifying 32
text editors 52
text files 52
textual cellview of half adder design 52
textual cellviews. See functional cellviews,
  compiling
top level design 77
transferring files to Import Files List Box 53

U
unbound components, symbol cellview for 82
unconstrained arrays 94

V
v 92
vector nets 87
vectors with noninteger indices 87
Verilog files 69
Verilog In tool 12
-version option 114, 116
VHDL
  case sensitivity 83
  constructs, structural nontranslatable 93
  defined 128
design files
  example of 40
  imported, compiling 32
design libraries
  adding to cds.lib file 19
  adding, using text editor 16
design process 12
design, intermediate representation of 12
designs importing to cells 41
text files 52
port modes 94
source files list of 31 names of 31 specifying 119
structural architectures converting to schematics 12
VHDL design conversion of 14
VHDL Import form 25, 69
(Ground) Value field 33
accessing using CIW menu options 23
using SKILL function 27
Add button 31
Compile VHDL Views After Import box 32
Compiler Options field 32
display 25
File Name field 31
Files List Box 31, 44
Ground Data Type field 33
Ground fields 33
Ground Net Name field 33
Ground Net Value field 33
Import Files List Box 31
import options 31
Import Structural Architectures As field 31
Maximum Number of Errors field 32
Net Name field 32
Overwrite Existing Views field 32
Power Data Type field 33
Power fields 32
Power Net Value field 32
Reference Libraries field 31
Remove button 31
Schematic Generation Options form 34
setting up 53, 58
Symbol View Name field 32
Target Library Name field 31
VHDL Work Library Name field 32
VHDL In commands getting help on 115
conversion limitations 14
Design Framework II 14
directory path 13
libraries needed 15
means available 12
output format, specifying 31
related products 12
standalone mode 14
starting in standalone mode 22
requirements for 14
ways of 13
with Design Framework II 23
version number, displaying 116
VHDL In output format
netlist 31
schematic 31
text file 31
VHDL In standalone options
+noxtrscl 117
-cdslib 117
-compile 117
-f 119
-help 115
-nocopyright 116
-param 119
-speedUp 118
syntax for 114
used with each other 114
-version 116
-work option 118
VHDL Interface product 12
VHDL to CDBA mapping 81
VHDL Tool Box form 26
VHDL Tool Box log file. See VHDL Tool Box status window
VHDL Tool Box status window 50, 62, 73, 103
command menu 103
using 103
VHDL WORK library
option for specifying 118
specifying name of 118
VHDL WORK Library Name field 32
vhdl.vhd file 52
vhdlActualPortFuncName property 89
vhdlArchitectureName property 82
vhdlAssignInstance property 92
vhdlAttributeDefList property 89
vhdlDataType property 87
vhdlFormalPortFuncName property 89
vhdlHilImport function 27
vhdlLibraryNames property 83
vhdlPackageComponents property 92
vhdlPackageNames property 83
vhdlPortType property 88
vhdlVecIndexType property 88
VHSIC, defined 128
view, defined 128
viewType, defined 128

W

windows
    CIW 14, 15
defined 128
    Library Manager 22, 51, 63
    Symbol Editor 64
    VHDL Tool Box status 50, 62, 73, 103
WORK library. See VHDL WORK library
-work option 118
workarea, TDM
    specifying 32
working library, defined 128
writeInRefLibs parameter 108
writeInRefLibs parameter 82