





CORE Generator™

System 2.1i

User Guide

 , XILINX, XACT, XC2064, XC3090, XC4005, XC-DS501, FPGA Archindry, NeoCAD, NeoCAD EPIC, NeoCAD PRISM, NeoROUTE, Plus Logic, Plustran, P+, Timing Wizard, and TRACE are registered trademarks of Xilinx, Inc.

 , all XC-prefix product designations, XACTstep, XACTstep Advanced, XACTstep Foundry, XACT-Floor-planner, XACT-Performance, XAPP, XAM, X-BLOX, X-BLOX plus, XChecker, XDM, XDS, XEPLD, XPP, XSI, Foundation Series, AllianceCORE, BITA, Configurable Logic Cell, CLC, Dual Block, FastCLK, FastCONNECT, FastFLASH, FastMap, HardWire, LCA, Logic Cell, LogiCore, LogiBLOX, LogicProfessor, MicroVia, PLUSASM, PowerGuide, PowerMaze, Select-RAM, SMARTswitch, TrueMap, UIM, VectorMaze, VersaBlock, VersaRing, and ZERO+ are trademarks of Xilinx, Inc. The Programmable Logic Company and The Programmable Gate Array Company are service marks of Xilinx, Inc.

All other trademarks are the property of their respective owners.

Xilinx does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx devices and products are protected under one or more of the following U.S. Patents: 4,642,487; 4,695,740; 4,706,216; 4,713,557; 4,746,822; 4,750,155; 4,758,985; 4,820,937; 4,821,233; 4,835,418; 4,853,626; 4,855,619; 4,855,669; 4,902,910; 4,940,909; 4,967,107; 5,012,135; 5,023,606; 5,028,821; 5,047,710; 5,068,603; 5,140,193; 5,148,390; 5,155,432; 5,166,858; 5,224,056; 5,243,238; 5,245,277; 5,267,187; 5,291,079; 5,295,090; 5,302,866; 5,319,252; 5,319,254; 5,321,704; 5,329,174; 5,329,181; 5,331,220; 5,331,226; 5,332,929; 5,337,255; 5,343,406; 5,349,248; 5,349,249; 5,349,250; 5,349,691; 5,357,153; 5,360,747; 5,361,229; 5,362,999; 5,365,125; 5,367,207; 5,386,154; 5,394,104; 5,399,924; 5,399,925; 5,410,189; 5,410,194; 5,414,377; 5,422,833; 5,426,378; 5,426,379; 5,430,687; 5,432,719; 5,448,181; 5,448,493; 5,450,021; 5,450,022; 5,453,706; 5,466,117; 5,469,003; 5,475,253; 5,477,414; 5,481,206; 5,483,478; 5,486,707; 5,486,776; 5,488,316; 5,489,858; 5,489,866; 5,491,353; 5,495,196; 5,498,979; 5,498,989; 5,499,192; 5,500,608; 5,500,609; 5,502,000; 5,502,440; RE 34,363, RE 34,444, and RE 34,808. Other U.S. and foreign patents pending. Xilinx, Inc. does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. Xilinx assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.

Copyright 1999 Xilinx, Inc. All Rights Reserved.

Table of Contents

About This Manual	vii
Manual Contents	vii
Conventions	ix
Typographical	ix
Online Document	x
Introduction	1-1
Overview	1-1
About the CORE Generator System	1-1
How to Obtain New and Updated COREs	1-4
System Requirements and Installation Information	1-5
Additional Resources	1-5
Online Documentation	1-5
Getting Started	2-1
Overview	2-1
CORE Generator Components	2-2
CORE Generator System Installation Requirements	2-2
Running CORE Generator System on Windows (95/98/NT)	2-2
Launching CORE Generator System on Workstations	2-2
Setup Files	2-3
coregen.prj	2-3
resources.lib	2-3
known.prj	2-4
coregen_<user_name>.prf	2-4
Sample Workstation preferences file:	2-6
Sample PC-based preferences file:	2-6
Inputs and Outputs	2-7
Project Management	2-9
Creating a New Project	2-9
Opening an Existing Project	2-10
Selecting Project Design Entry Options	2-11
Selecting Vendor Design Entry Options	2-11
Selecting Behavioral Simulation Options	2-12
Selecting Target Xilinx FPGA Family Options	2-12

Complete New Project Creation	2-12
Changing Project Design Entry Options	2-13
Setting Web Browser Path	2-13
Setting AcroRead Path	2-13
CORE Generator System Troubleshooting	2-14
General Hints:	2-14
Where to get help with general CORE Generator problems:	2-15
Additional Resources:	2-15
Xilinx CORE Generator System	2-15
AllianceCORE	2-15
Email Support	2-16
Web Support	2-16
FTP	2-16
Using the CORE Generator™ System	3-1
Overview	3-1
Using the CORE Browser	3-1
Accessing CORE Data Sheets	3-3
Customizing a CORE	3-4
Illegal or Invalid Values	3-6
.COE Files	3-6
Command Files	3-11
coregen.ini/coregen_<user_name>.ini	3-11
User-generated Command Files	3-11
.XCO Files	3-12
coregen.log	3-13
Valid CORE Generator Commands	3-14
Global Properties	3-14
Project Properties	3-15
Generating COREs in Batch Mode	3-15
Updating COREs in the CORE Generator System	3-17
Version Handling in the CORE Generator System	3-17
Downloading New COREs	3-17
CORE Version Update in an Existing Project	3-18
CORE Generator Design Flows	4-1
Overview	4-1
CORE Generator Design Flow Basics	4-2

Viewlogic Design Flow	4-3
Create a directory for a Viewlogic project	4-3
Example	4-3
Set up project libraries	4-3
Set the Output Format	4-4
Set Project Path and Viewlogic Library Alias	4-4
Select Desired Module	4-5
Output Files	4-5
Load Symbol in Schematic Editor	4-6
Example	4-6
Foundation Design Flow	4-6
Mentor Design Flow	4-7
Cadence Design Flow	4-7
HDL Design Flows	4-7
Behavioral Model Delivery in the CORE Generator System	4-8
get_models	4-9
Syntax	4-9
Required Parameters	4-9
Inputs	4-10
Outputs	4-10
.VEO (Verilog Instantiation Template)	4-10
.VHO (VHDL Instantiation Template)	4-12
Sample .VHO file for an 8-bit adder	4-12
CORE Generator Verilog Flow	4-15
Synthesis	4-15
Simulation	4-15
Basic Steps in the Verilog Design Flow	4-15
Module Generation	4-15
Behavioral Simulation	4-15
Module Generation	4-16
Specify the Design Entry, Vendor, and Behavioral Simulation settings for the project.	4-16
Behavioral Simulation	4-16
Preparing for Verilog Behavioral Simulation	4-16
Instantiate the core (same procedure for all simulators and synthesis tools)	4-17

Cadence Verilog-XL and MTI ModelSim/VLOG	4-18
Example	4-18
Comment out the instantiation template in the .VEO file.	4-18
Copy the .VEO file to <module_name>.v	4-18
Connect the core to the parent design by editing the module connections.	4-18
Example	4-18
VERILOG instantiation template file: myadder8.veo	4-19
VERILOG parent design file: myadder8_top.v	4-21
Create The Testbench	4-23
Perform the Behavioral Simulation	4-24
MTI ModelSIM	4-24
Cadence Verilog-XL	4-25
Synthesis	4-25
Synopsys FPGA Compiler	4-25
Synopsys FPGA Express	4-25
Synplicity Synplify	4-25
Exemplar Leonardo	4-26
Write out the implementation netlist for the synthesized design	4-26
Synopsys FPGA Compiler	4-27
Synopsys FPGA Express	4-27
Synplicity Synplify	4-27
Exemplar	4-27
Implementation	4-27
CORE Generator VHDL Flow	4-28
Synthesis	4-28
Simulation	4-28
Basic Steps	4-28
Module Generation	4-29
Synopsys FPGA Express and FPGA Compiler	4-30
Synplicity Synplify	4-30
Exemplar Leonardo	4-30
Behavioral Simulation	4-30
Prepare for Simulation	4-30
VHDL Model Extraction	4-30

Example	4-31
Simulator-specific processing steps	4-31
Create the XilinxCoreLib library	4-32
MTI ModelSim/VHDL	4-32
MTI ModelSim/VHDL	4-32
Analyze the behavioral models	4-32
MTI ModelSim/VHDL	4-33
Instantiate the module (same procedure for all simulators)	4-33
Connect the core to the parent design by editing the instantiation block	4-34
Example	4-35
VHDL template file: myadder8.vho	4-35
VHDL parent design file: myadder8_top.vhd	4-38
Create the Testbench	4-40
VHDL Testbench file: testbench.vhd	4-40
Design Simulation	4-42
Analyze the parent design and testbench file	4-42
MTI ModelSim	4-42
Invoke the simulator	4-42
Synthesis and Implementation	4-42
Synthesize the design using black-box methodology	4-42
Synopsys FPGA Compiler	4-43
Synopsys FPGA Express	4-43
Synplicity Synplify v5.1.2 and later	4-43
Exemplar Leonardo (v1998.2)	4-44
Write out the implementation netlist for the synthesized design	4-44
Synopsys FPGA Compiler	4-44
Synopsys FPGA Express	4-44
Synplicity Synplify v5.1.2 and later	4-44
Exemplar Leonardo (v1998.2)	4-45
Implement the Design	4-45

Table of Contents

About This Manual

This manual describes the Xilinx CORE Generator™ System, a tool used for parameterizing cores optimized for Xilinx FPGAs.

The CORE Generator System User Guide is available online and can be read using the Acrobat Reader. This manual can be launched as follows: From the pulldown menu area of the “CORE Generator (Main Menu)” window, select HELP--> Online Documentation.

Before using this manual, you should be familiar with the operations that are common to all Xilinx software tools. These operations are covered in the *Quick Start Guide* of the Xilinx software documentation.

Manual Contents

This manual covers the following topics:

- Introduction (Overview)
- Getting Started
- How to Use the CORE Generator System
- Design Flows in the CORE Generator System

Conventions

This manual uses the following typographical and online document conventions. An example illustrates each typographical convention.

Typographical

The following conventions are used for all documents.

- `Courier font` indicates messages, prompts, and program files that the system displays.

```
speed grade: -100
```

- **Courier bold** indicates literal commands that you enter in a syntactical statement. However, braces “{ }” in Courier bold are not literal and square brackets “[]” in Courier bold are literal only in the case of bus specifications, such as bus [7:0].

```
rpt_del_net=
```

Courier bold also indicates commands that you select from a menu.

```
File → Open
```

- *Italic font* denotes the following items.
 - Variables in a syntax statement for which you must supply values

```
edif2ngd design_name
```

- References to other manuals

See the *Development System Reference Guide* for more information.

- Emphasis in text

If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.

- Square brackets “[]” indicate an optional entry or parameter. However, in bus specifications, such as bus [7:0], they are mandatory.

```
edif2ngd [option_name] design_name
```

- Braces “{ }” enclose a list of items from which you must choose one or more.

```
lowpwr = {on | off}
```

- A vertical bar “|” separates items in a list of choices.

```
lowpwr = {on | off}
```

- A vertical ellipsis indicates repetitive material that has been omitted.

```
IOB #1: Name = QOUT'  
IOB #2: Name = CLKIN'  
.  
.  
.
```

- A horizontal ellipsis “...” indicates that an item can be repeated one or more times.

```
allow block block_name loc1 loc2 . . . locn;
```

Online Document

The following conventions are used for online documents.

- Red-underlined text indicates an interbook link, which is a cross-reference to another book. Click the red-underlined text to open the specified cross-reference.
- Blue-underlined text indicates an intrabook link, which is a cross-reference within a book. Click the blue-underlined text to open the specified cross-reference.

Introduction

Overview

This chapter introduces the Xilinx CORE Generator™ System, an easy to use design tool that delivers parameterizable COREs optimized for Xilinx FPGAs.

The following topics are included in this chapter:

- About the CORE Generator System
- System Requirements and Installation Information
- How to Obtain New and Updated COREs

About the CORE Generator System

The CORE Generator System's main Graphical User Interface (GUI) allows central access to COREs, data sheets, variable options, and help functions.

The Xilinx CORE Generator System provides the user with a catalog of ready-made functions ranging in complexity from simple arithmetic operators such as adders, accumulators and multipliers, to system-level building blocks including filters, transforms and memories.

Note: The words **function** and **CORE** are used interchangeably in this guide to mean a design entity like a multiplier or FIR filter which the CORE Generator System can generate for the designer.

COREs are organized by type into folders that expand or contract on demand. Detailed information on each CORE is contained in a specification or data sheet which can be accessed by clicking on the Datasheet button in the CORE specification window, or by clicking

on the Datasheet icon in the main CORE Generator application toolbar. This launches the Adobe Acrobat Reader and calls up the datasheet for the selected CORE. Datasheets include:

- Functional information
- Area and performance data
- Pinouts and interface signal names
- Details of how to use the CORE in an application

The CORE Generator System can customize a generic functional building block such as a FIR filter or a multiplier to meet the needs of your application and simultaneously delivers high levels of performance and area efficiency. This is accomplished by the use of Xilinx's CORE-friendly FPGA architectures and by the application of Xilinx Smart-IP™ technology.

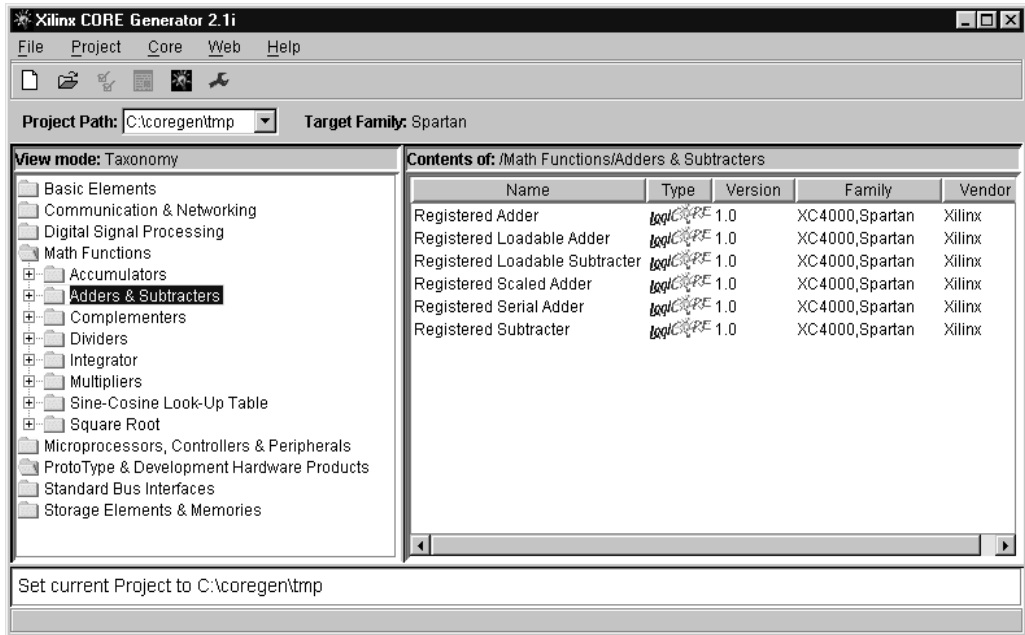


Figure 1-1 CORE Generator System Window

Smart-IP™ technology leverages:

- Xilinx FPGA architectural advantages such as look-up tables (LUTs), distributed RAM, segmented routing and floorplanning information
- Relative location constraints and expert logic mapping to optimize performance of a given CORE instance in a given Xilinx FPGA architecture

Smart-IP™ technology delivers:

- Physical layout optimized for high performance
- Predictable performance and resource utilization
- Reduced power requirements through compact design and interconnect minimization
- Performance independent of device size
- Ability to use multiple COREs without deterioration of performance
- Reduced compile time over competing architectures.

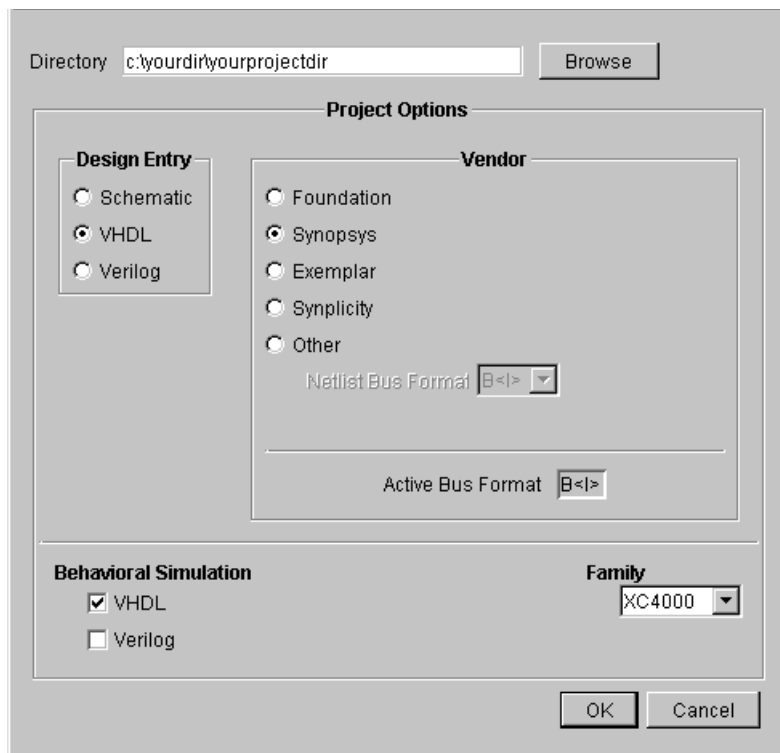
Parameterization provides the ability to generate COREs which meet design flexibility needs and which meet design size constraints.

For each CORE the CORE Generator System delivers:

- A customized EDIF netlist
- In addition the user had the option of generating support files with Verilog or VHDL behavioral simulation models
- Instantiation templates
- Foundation or Viewlogic schematic symbols

This makes it easy for you to integrate the COREs into your preferred design environment using the CORE Generator System.

High performance, area efficiency, and flexibility delivered by the CORE Generator System's diverse library of complex building blocks make the design of systems-on-a-chip simpler and faster than ever before.



1-2 F

How to Obtain New and Updated COREs

New COREs can be downloaded from the Xilinx web site and easily added to the CORE Generator System.

Bookmark: <http://www.xilinx.com/products/logicore/coregen>

Xilinx encourages you to check the CORE Generator System web page before starting a new design to verify that you have the latest version of each CORE and CORE datasheet; the CORE Generator web page also provides a current list of new and updated COREs that may be useful in your design.

System Requirements and Installation Information

See the 2.1i Release Notes for information on system requirements and installation instructions for the CORE Generator System.

Adobe Acrobat v 3.0 or later is needed to launch and view the COREs datasheets.

The CORE Generator System interface to Viewlogic requires that both the Viewlogic and the Xilinx Implementation Tools be set up on your system.

Additional Resources

Online Documentation

For detailed help on how to operate the Xilinx CORE Generator System, you can select **Help->Topics** from the **Help** menu at the top of the CORE Generator main window. Refer to the appropriate section for your design flow.

For an overview of the supported design flows, refer to chapter 4, *CORE Generator Design Flow*, in this manual

select **Help->Topics** under the Help Menu and select *User Guide* to access the online user guide. This Guide provides descriptions and step-by-step instructions on using the Xilinx CORE Generator System.

Getting Started

Overview

This chapter describes the various elements of the CORE Generator™ System. The user must have this information before creating successful designs targeted to the Xilinx FPGAs, using one or more of the COREs offered in the CORE Generator System.

The elements of the CORE Generator System included in this chapter are:

- Components
- System Installation Requirements
- Setup Files
- Inputs and Outputs
- Project Management
- Troubleshooting

CORE Generator Components

The Xilinx CORE Generator System consists of three distinct products:

- The Acrobat Reader Application
- The CORE Generator Application
- JAVA Run-time Support

CORE Generator System Installation Requirements

For detailed information on CORE Generator installation on the PC and on the Unix Workstation refer to the *Xilinx Alliance/Foundation Quick Start Guide (version 2.1i)*.

Running CORE Generator System on Windows (95/98/NT)

To start the CORE Generator System from the Windows menu, click on the Windows Start Menu button, select:

```
'Programs'--> 'Xilinx Alliance Series 2.1i or  
Foundation Series 2.1i-->'Accessories'--  
> 'COREGen'
```

Launching CORE Generator System on Workstations

Make sure your environment is set up to run the Xilinx software as specified in the *Xilinx Alliance/Foundation Quick Start Guide (v2.1i)*. The two required settings are: the XILINX variable, which should be set to your Xilinx installation directory, and your PATH variable, which should be set to \$XILINX/bin/<platform>, where "platform" is either "sol" or "hp".

Type **coregen** to start the CORE Generator System.

Setup Files

This section describes in some detail the setup files in the CORE Generator System. The setup files are required to properly configure your CORE Generator session.

- coregen.prj
- resources.lib
- known.prj
- coregen_<user_name>.prf

coregen.prj

coregen.prj is the CORE Generator "project file". The coregen.prj file is automatically created whenever you create a new project. It contains a record of project-specific property settings, information on versions of the COREs available to the project, and user-specified output files. A valid CORE Generator project directory must contain a coregen.prj file.

The information in the coregen.prj file includes a list of all the IP COREs and versions that are available to the project, as well as the version of every CORE actually used in the project.

coregen.prj is a configuration file which is created, read, and modified by the CORE Generator System for project management purposes and should not be altered by the user.

resources.lib

Resources.lib is the cores database file located in `$XILINX/coregen/ip`. It is generated in the `$XILINX/coregen/ip` directory the first time a user starts up a given installation of the CORE Generator software. This database is built by scanning the COREs installed in `$XILINX/coregen/ip` and is used by the CORE Generator System to identify and locate all COREs that are present in a CORE Generator software installation.

The database recorded in resources.lib is also updated whenever new COREs are added into the IP repository. An update of the resources.lib database file can be forced by placing a file named `<container_name>.upd` in the `$XILINX/coregen/ip`

Note that since the COREs database is built at startup time, new versions of COREs (that are deposited into the repository while CORE Generator System is running) will not be recognized until after the CORE Generator System is restarted.

known.prj

known.prj, located in `$XILINX/coregen/preferences`, contains a listing of fully qualified paths to all known user projects. These projects are listed in the known projects listing drop-down in the CORE Generator GUI. When multiple users are using the same installation of the CORE Generator System, this list will include all known projects for all those users.

Individual projects can be deleted from the known projects listing by manually deleting the corresponding line in the known.prj file.

coregen_<user_name>.prf

coregen_<user_name>.prf is the Xilinx CORE Generator preferences file. This is an ASCII option settings file that allows the user to customize various aspects of the CORE Generator GUI, and certain kinds of runtime behavior. This file is user-specific and consists of a mix of comment lines and property specification lines. Comment lines begin with the "#" (octothorpe) character and designate a line which is ignored when the file is read by the CORE Generator System. The format for a line specifying a property in the preference file is:

```
PropertyName = Value
```

Example: AlwaysOpenLastProject = true

Each Property Name represents a particular property within the Xilinx CORE Generator System, and the corresponding Value field is the value to be applied to that property. Preference settings are stored for each user in the directory:

```
$XILINX/coregen/preferences
```

The name of a given user's preference file is:

```
coregen_<user_name>.prf
```

where `<user_name>` is replaced by the name of the user. The `<user_name>` part of the file name is based on the computer name of the user as defined in their Windows settings or UNIX login name.

The first time a specific user starts up the CORE Generator System, no preference file exists for that user. The preference file, `coregen_<user_name>.prf`, for a particular user is created the first time the user exits out of CORE Generator System. The file is automatically written to `$XILINX/coregen/preferences` when the user exits the CORE Generator application, based on settings they have specified during a project session. This file can also be created manually by the user as a text file.

The CORE Generator System looks for preference files in the `$XILINX/coregen/preferences` directory.

During start-up, and after any optional `coregen.ini` file is read (Workstations only), CORE Generator System searches the preferences directory for a `coregen_user_name.prf` file. If this file is found, it is loaded and all preferences contained in it override the default CORE Generator System preference settings. If no preference file is found for the user (as in the case of a first-time user), the various preference values take on their hardcoded default values.

The list of supported preference file properties is shown below.

Property Name	Value	Definition
AlwaysOpenLastProject	false	Do not start CORE Generator System in the last active project.
	true	Always start CORE Generator System in the last active project.
OverwriteFiles	false	Prompt user before overwriting design files during elaboration.
	true	Automatically overwrite design files during elaboration.

Property Name	Value	Definition
Browser	<path_to _web_br owser>	Fully qualified path to the user's web browser.
Viewer	<path_to _pdf_vie wer>	Fully qualified path to the user's Acrobat PDF viewer.

Sample Workstation preferences file:

```
#Coregen preferences
#Fri Apr 23 13:46:57 PDT 1999
lastproject=/home/myprojects/fir_filter
alwaysopenlastproject = false
overwritefiles = true
browser = /usr/bin/netscape
viewer = /usr/local/bin/acroread
```

Sample PC-based preferences file:

```
#Coregen preferences
#Fri Apr 23 13:46:57 PDT 1999
lastproject=\\home\\myprojects\\fir_filter
alwaysopenlastproject = false
overwritefiles = true
browser = c:\\program files\\netscape\\communi-
cator\\program\\netscape.exe
viewer = c:\\program
files\\acrobat3\\reader\\acrord32.exe
```

Note: The users should not edit their preferences file directly.

Inputs and Outputs

This section lists the input files used by, and the output files generated by the CORE Generator System.

Input	Description
.COE	ASCII data file. Defines the coefficient values for FIR Filters and initialization values for Memory modules. See \$XILINX/coregen/data for sample .COE files.
.XCO	CORE Generator file containing the parameters used to specify the parameters for regenerating a CORE and can also be used as a logfile to determine the settings used to generate a particular CORE. This file is also generated by the CORE Generator System along with any CORE that it creates in the current project directory. For details on the .xco file refer to the <i>Using the CORE Generator System</i> chapter in this user guide.

Output	Description
.EDN	EDIF Implementation Netlist for the CORE. Describes how the CORE is implemented and is used as input to the Xilinx Implementation Tools.
.VEO	Verilog Template file. The components in this file can be used as a guide to creating the core's Verilog instantiation and its Verilog behavioral netlist. For more details refer to the section on get_models in Chapter 4 of this manual.
.VHO	VHDL Template file. The components in this file can be used as a guide to creating the CORE's VHDL instantiation and its VHDL behavioral simulation reference. For more details, refer to the section on get_models in Chapter 4 of this manual.

Output	Description
.EDN	EDIF Implementation Netlist for the CORE. Describes how the CORE is implemented and is used as input to the Xilinx Implementation Tools.
.XCO	CORE Generator file containing the parameters used for generating a customized CORE. Maybe created manually by the user, but this file is also generated by the CORE Generator System, when first creating a CORE.
.MIF	Memory Initialization File which is automatically generated by the CORE Generator System for Virtex Block RAM modules, only when an HDL simulation flow is specified. A MIF data file is used to support HDL functional simulation of Block RAM modules. To generate a MIF file you must direct CORE Generator System to generate an EDIF Implementation Netlist and specify either a Verilog or a VHDL Behavioral Simulation output.
vllink.log	Log file created by the VLLINK script when a Viewlogic Schematic Design Flow is selected from the Project Option menu. This file records the operations and status of the various function calls used to create a Viewlogic symbol and WIR file for core simulation.

Project Management

This section describes in some detail the functions performed by the user in initiating, designing, and maintaining core designs in Xilinx CORE Generator System's GUI environment. The functions included in this section are: *Creating a New Project*, *Selecting an Existing Project*, *Selecting Project Design Entry Options*, *Selecting Behavioral Simulation Options*, *Selecting Target Family Options*, *Finish New Project Creation*, *Changing to Another Project*, *Changing Project Design Entry Options*, *Setting Web Browser Path*, and *Setting AcroRead Path*.

Note: The Xilinx CORE Generator System is designed to operate within the directory structure of your selected design entry environment. Consequently, the CORE Generator System does not create directories and hence, make sure the directory exists before browsing to it.

Creating a New Project

When you first start up the Xilinx CORE Generator System, you will be prompted to create a new project with the "Getting Started" project dialog. The "Create a New Project" radio button should be selected. If it is not, please select it now. You can type the path to the new project directory into the "directory" text field or you may click on the "browse" button and navigate to it.

Specify your CAE Vendor (Foundation, Viewlogic, Cadence, Mentor, or Other). If you select "Other", you must also specify the required Netlist Bus Format for individual bus bits ("B<I>", "B(I)", or "BI", where "B" represents the name of the bus, and "I" represents the bus index. Selecting any of the other vendors will automatically set the Netlist Bus Format setting to the correct value for that vendor.

Note: Due to a browser limitation, once you have "browsed to the project directory, you must specify a file by the name of `coregen.prj` for the project file. For "new" projects this file will not exist, but will be created when you click "OK".

For example, (for PCs): `c:\designs\newproject\coregen.prj`

For workstations: `../designs/newproject/coregen.prj`

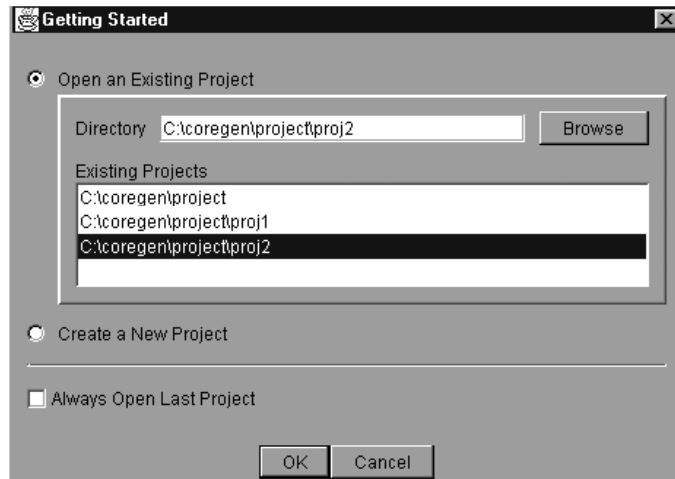


Figure 2-1 Getting Started Project Dialog Window

Opening an Existing Project

The Xilinx CORE Generator System 2.1i maintains a list of valid projects. When the CORE Generator System is started you are provided with a list of these valid projects. In a shared-user environment in which multiple users access the same installation of the CORE Generator System this would include projects belonging to all users using that installation. You may select a project from the list in the "Getting Started" dialog window. Select the CORE Generator project that you want to work in and click **OK**.

Within the "Getting Started" project dialog you may also place a check mark for the "Always Open Last Project" check box. This will cause the CORE Generator System to bypass the "Getting Started" dialog window and always open the last project that you were working on.

Note: To restore the launching of the "Getting Started" project dialog window, you must deselect the corresponding check box in the "Open Project" dialog window.

Selecting Project Design Entry Options

From the Getting Started dialog window select the Create a New Project button and this will bring up the "New Project" dialog window. You will now need to select the top level "Design Entry" flow that you will be using. The Xilinx CORE Generator System supports "Schematic", "VHDL", and "Verilog" top level design entry flows. Select one of these design entry flows.

Selecting Vendor Design Entry Options

From the "Project Options" window select the Design Entry Vendor that you will be using. Based on your selection, the output EDIF netlist will contain the appropriate bus delimiter symbol for the EDIF netlist for the module: "()", "<>", "[]"; none (This will allow the generated EDIF netlist port to match the port references in the EDIF

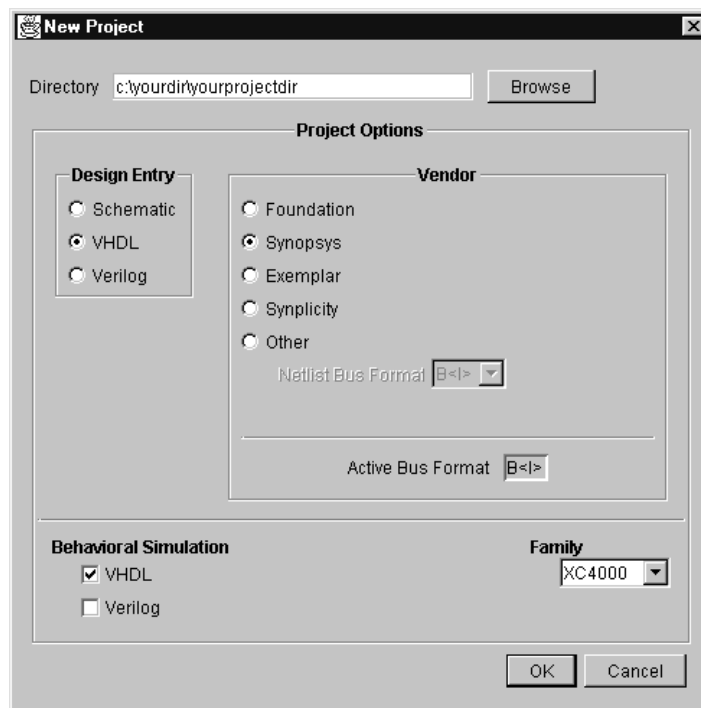


Figure 2-2 Project Options Dialog Window

netlist for your top level design, for your module, the HDL instantiation templates or schematic symbol port and pin names.

If the Design Entry Vendor you are using is not listed, select "Other" as your Vendor. You will also need to specify the netlist bus format that your HDL or Schematic tools support.

Selecting Behavioral Simulation Options

The Behavioral Simulation templates that the Xilinx CORE Generator System creates are dependent on the Behavioral Simulation options (Verilog and/or VHDL) flow that you select. If you would like additional behavioral simulation templates for other simulation flows, you may select them. For example, if you have selected VHDL as your Behavioral Simulation option, a VHDL simulation template file will be generated with the name of `<module_name>.vho`. For example, if you would like a Verilog instantiation template and/or a Verilog behavioral simulation template, you will also need to place a check mark in the "Verilog Behavioral Simulation" check box. This will cause a `<module_name>.veo` file to be created as well.

Selecting Target Xilinx FPGA Family Options

The Xilinx CORE Generator System tailors the COREs it generates to the selected "Target Family" setting. Any COREs that are generated are optimized to this selected Xilinx architecture and will not work if integrated into a design targeted for a different Xilinx FPGA family. For example, COREs that were targeted to the Spartan architecture, when they were generated, will not work if placed in a Virtex design. Select the "Target Family" based on the Xilinx architecture that you will be targeting. If you change architectures you will need to regenerate any COREs you have already created.

Complete New Project Creation

After you have selected all the project options, click **OK**. The Xilinx CORE Generator System will now initialize the new project. A `coregen.prj` file is written to the new project. The `coregen.prj` file contains a record of all installed COREs and their available versions. A link to the new project is also written to the `known.prj` file found in the `$XILINX/coregen/preferences` directory.

Note: The initialization of the new project may take several seconds. Also the listing of COREs available in the CORE Generator System will not display until you specify a valid CORE Generator project.

Changing Project Design Entry Options

You may change the Project Design Entry Options by selecting **Project-->Project Options** menu. This will launch a "Project Options" dialog. You may change any of the "Design Entry", "Vendor", "Behavioral Simulation" and "Family" options. When you have finished modifying these options, click **OK**.

You should be aware that changing the project options will only affect new COREs that you generate. Any COREs that you have created before making the project changes will still reflect the old options. You will need to regenerate any COREs that need to inherit the new project options.

Setting Web Browser Path

The Xilinx CORE Generator System is able to link to sites on the Web. You may click to the Xilinx CORE Generator System Web page or the Xilinx support.xilinx.com site. You can also link to the AllianceCORE partner web sites. When the Xilinx CORE Generator System attempts to link to the Web the first time, it will look for your default browser.

The Xilinx CORE Generator System will first attempt to locate your default browser from the registry (on the PC) or from your path (on Workstations). If it cannot determine the default browser, you will be prompted for the location. A "Choose Browser" dialog will launch and allow you to provide the path to your web browser if it has not been previously set.

The path to your web browser can also be set by editing \$XILINX/coregen/preferences/coregen-<user_name>.prf and setting the "Browser" property value to the fully qualified path to your web browser.

Setting AcroRead Path

The Xilinx CORE Generator System provides all core datasheets in Adobe Acrobat PDF format. When the Xilinx CORE Generator System attempts to link to the Web the first time, it will look for your

default PDF Acrobat Reader. The Xilinx CORE Generator System will first attempt to locate your default PDF Acrobat Reader from the registry (on the PC) or from your path (on Workstations). If it cannot determine the default PDF Acrobat Reader, you will be prompted for the location. A `Choose PDF Reader` dialog will launch and allow you to provide the path to your PDF Acrobat Reader.

The path to your Acrobat Reader can also be set by editing the "Viewer" property value in `$XILINX/coregen/preferences/coregen_<user_name>.prf`. Refer to the section on Setup Files, earlier in this chapter for more information on setting the "Viewer" property value.

CORE Generator System Troubleshooting

General Hints:

- Check the `coregen.log` file and `<module_name>.xco` file for diagnostic information.
- If your `coregen.prj` project information file becomes corrupted, delete it and recreate the project in that directory by selecting the New Project option in CORE Generator System. One symptom of a corrupted `coregen.prj` is errors from the CORE Generator System during startup about missing modules.
- Check that write permissions are open for the directories `$XILINX/coregen/ip`, `$XILINX/coregen/preferences`, and `$XILINX/coregen/tmp`. These directories must be writeable for proper functioning of the CORE Generator System. Problems creating the `resources.lib` file may be associated with write permissions or lack of available disk space on the `$XILINX/coregen/ip` directory.
- `LD_LIBRARY` errors (workstations only): Verify that `LD_LIBRARY_PATH` includes the path to `$XILINX/bin/<platform>`

To debug startup problems, edit `coregen.bat` and add `-v` (verbose mode) and `-d` (debug mode) options to the `java.exe` command line in `coregen.bat`. The `-v` option will cause CORE Generator System to display a detailed report of all data files being loaded and miscellaneous operations. The `-d` option will cause CORE Generator System to report specific debug-related information.

Where to get help with general CORE Generator problems:

- Check here first: <http://www.support.xilinx.com/support/techsup/tappinfo.htm> for general information on contacting Xilinx support.
- Use our web-based search engine to search the Xilinx Answers Database <http://www.support.xilinx.com/support/searchtd.htm>. This database contains information on all known problems with Xilinx hardware and software. Xilinx Applications Engineers add to this knowledge base daily.
- Check the Updates & Files area, given below, to make sure your software is current.

http://www.support.xilinx.com/support/techsup/sw_updates

Additional Resources:

Xilinx CORE Generator System

Xilinx CORE Generator web page:

<http://www.xilinx.com/products/logicore/coregen>

Check here for the latest news on the CORE Generator System, including announcements about new modules.

CORE Generator & IP Modules Technical Tips web page:

<http://www.xilinx.com/support/techsup/journals/coregen>

AllianceCORE

Modules:

Contact the appropriate third party AllianceCORE provider as indicated on the CORE Generator datasheet for that module.

Email Support

For CORE Generator System specific questions not covered in the Xilinx Answers Database:

`coregen@xilinx.com`

For DSP application specific questions:

`dsp@xilinx.com`

For AllianceCORE Partner questions:

`alliancecore@xilinx.com`

Web Support

For CORE Generator product information:

`http://www.xilinx.com/products/logicore/coregen`

For information on cores:

`http://www.xilinx.com/ipcenter`

FTP

Miscellaneous application notes and design files:

`http://www.xilinx.com/support/techsup/ftp/apps.htm`

Using the CORE Generator™ System

Overview

This chapter explains the major functions performed by the designer when using the CORE Generator System. The functions included in this chapter are:

- How to use the CORE browser
- How to access CORE datasheets
- How to customize a CORE
- How to generate COREs in batch mode
- How to update the COREs

Using the CORE Browser

The main view of the CORE Generator System is the CORE browser.

COREs that fall into particular application categories are grouped into folders to assist you in locating the CORE appropriate to your needs. The left hand pane of the CORE browser allows you to browse through these folders. To select a folder, click once on the folder name in the left hand pane. To expand a folder, double click on the folder icon to the left of the folder name. The folder will expand to reveal more folders. A folder may be closed by double clicking on the open folder icon. Some folders have a + icon or a - icon to their left which can also be used to open or close them respectively by a single click on the icon.

The COREs in the selected folder are displayed in the right hand pane of the CORE browser. COREs are listed by name and also have type, version, family and vendor information displayed in columns. The

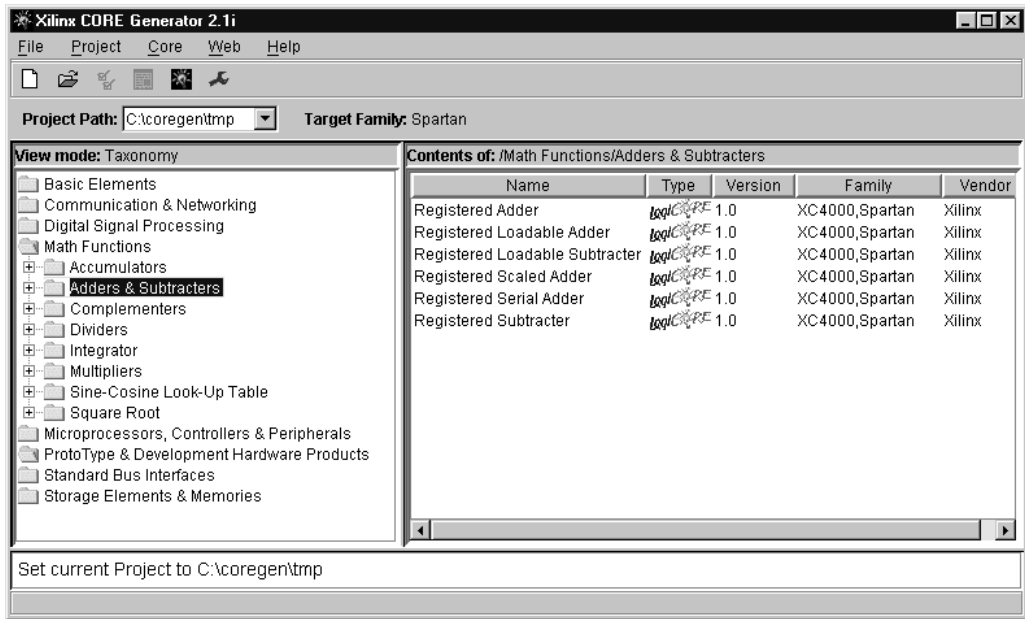


Figure 3-1 Main CORE Generator Browser Window

size of each column can be altered by pointing the mouse at the separator between the column headings until the cursor changes to

↔ icon. Click and hold the left mouse button (clicking slightly to the left of the separator selects the column on the left and clicking slightly to the right selects the column on the right of the separator) and then move the mouse horizontally to resize the selected column. Release the mouse button when the desired column width is achieved. The column ordering can also be modified by clicking and holding the left mouse button while pointing at the category label, then dragging the label to the desired position by moving the mouse horizontally. Releasing the mouse button deposits the label at its new position.

The status pane at the bottom of the CORE browser window displays the results of actions and, displays appropriate messages if any errors or warnings occur.

The size of each of the panes can be adjusted by pointing at the vertical or horizontal bar separating the panes until the cursor changes to a ↑ icon (for vertical bars) or ↔ icon (for horizontal bars). Clicking and holding the left mouse button allows the panes to be

resized when the mouse is moved. Release the mouse button when the desired pane layout is achieved.

All panes will have vertical and or horizontal scroll bars that allow navigation if the information displayed in the pane is larger than the current pane size.

Accessing CORE Data Sheets

A CORE can be selected by clicking on the name of the CORE in the right pane. Once a CORE has been selected the data sheet can be viewed by clicking on the Data Sheet button on the CORE browser toolbar or by selecting the **Core -->Datasheet** option from the pull-down menus. Both of these actions will launch the Acrobat reader to display the data sheet.

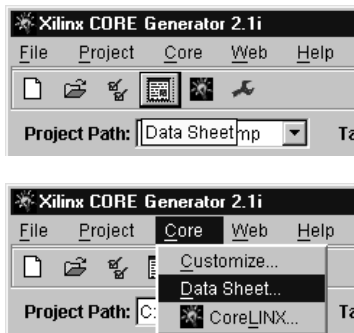


Figure 3-2 CORE Datasheet Buttons

Customizing a CORE

Most COREs have a parameterization window. The parameterization window shown below was launched by navigating to the CORE Generator's 'Registered Loadable Adder' module.

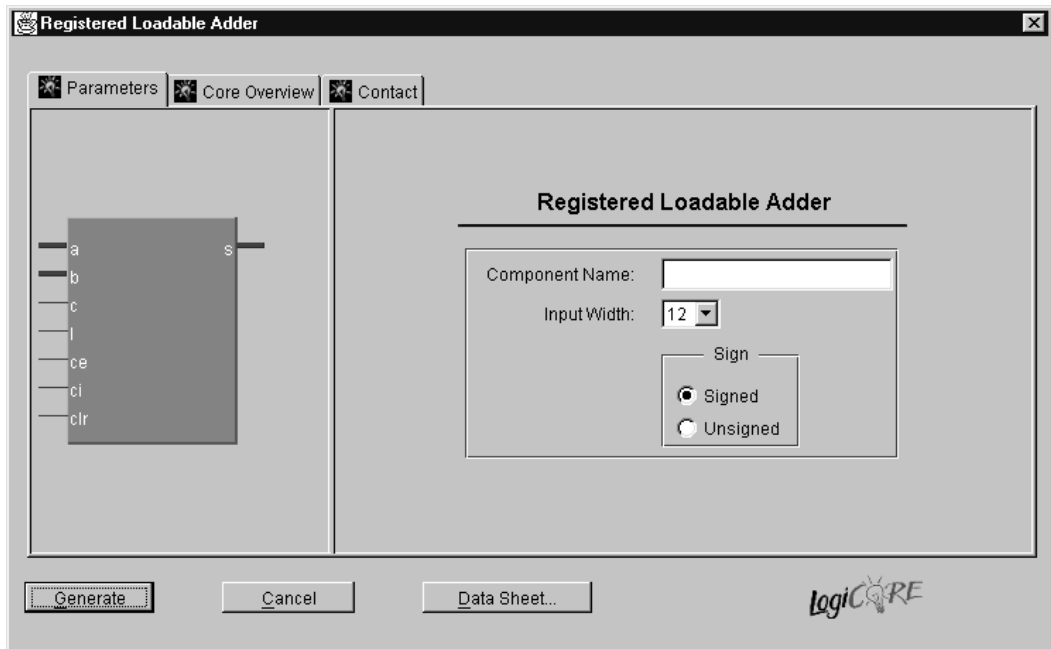


Figure 3-3 Registered Loadable Adder Module

The parameterization window for a CORE is displayed by double clicking on the CORE in the right pane of the CORE browser, clicking on the *Customize* button on the CORE browser toolbar, or selecting:

Core -> Customize from the pull down menus.

While the parameterization windows will be unique for each CORE, there are some characteristics that are common to all modules.

The default tab that is displayed when the parameterization window is displayed is the Parameter screen. A CORE Overview screen is available as is a Contact Information screen. These tabs are accessed

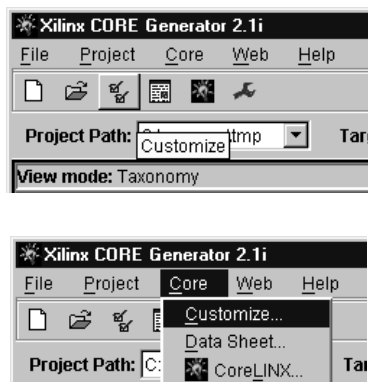


Figure 3-4 CORE Customize Buttons

by clicking on the appropriate tab at the top of the parameterization window.

All modules have a `Component Name` field which allows you to assign a name to the CORE that you create. Files that the CORE Generator System creates for a particular CORE will have a root file-name that matches the Component Name. You should note that component names have the following restrictions:

- Must begin with an alpha character: **a - z**
- No uppercase letters
- May include (after the first character): **0 - 9**, **_**(underscore)
- No extensions

The **Generate**, **Cancel** and **Data Sheet ...** buttons are also common to all parameterization windows. Assuming there are no problems with any of the parameters that have been specified, pressing **Generate** will cause the CORE Generator System to create files for the requested CORE of the requested types. Pressing **Cancel** will return you to the CORE browser window without generating any files. Pressing **Data Sheet ...** will invoke Adobe Acrobat to display the data sheet for the module being parameterized.

For information about a specific CORE's parameterization window, such as upper and lower limits for certain fields, see the CORE's data sheet.

Illegal or Invalid Values

All parameterization windows flag illegal or invalid data in the same fashion. The affected field is highlighted in red until the problem is corrected. If the reason why a field is highlighted is not obvious, or if the explanation in the log window is not clear, a more detailed explanation can usually be obtained by pressing the **Generate** button.

.COE Files

Some COREs require a significant amount of information to fully specify their behavior. COREs of this type (e.g. PDA FIR, SDA FIR, XC4000 RAM and ROM, and Virtex Block RAM) usually require multiple coefficients or initialization values. To specify the values more conveniently, you can load a .COE file using the **Load Coefficients...** button in the parameterization window.

The **Load Coefficients...** button can be seen in the screen shot of the PDA FIR Filter parameterization window Figure 3-5.

Additional information about the requirements for a CORE's .COE file can be found in that CORE's data sheet.

The general form for a .COE file is the following:

Keyword =Value ; Optional Comment

Keyword =Value ; Optional Comment

CoefData =Data_Value, Data_Value, ...;

CoefData is the keyword used for filters to indicate that the data that follows comprises the coefficients for the filter. For distributed memories the keyword MemData is used instead and for Virtex Block memories the keyword is

MEMORY_INITIALIZATION_VECTOR

Note: Any text after a semicolon is treated as a comment and will be ignored.

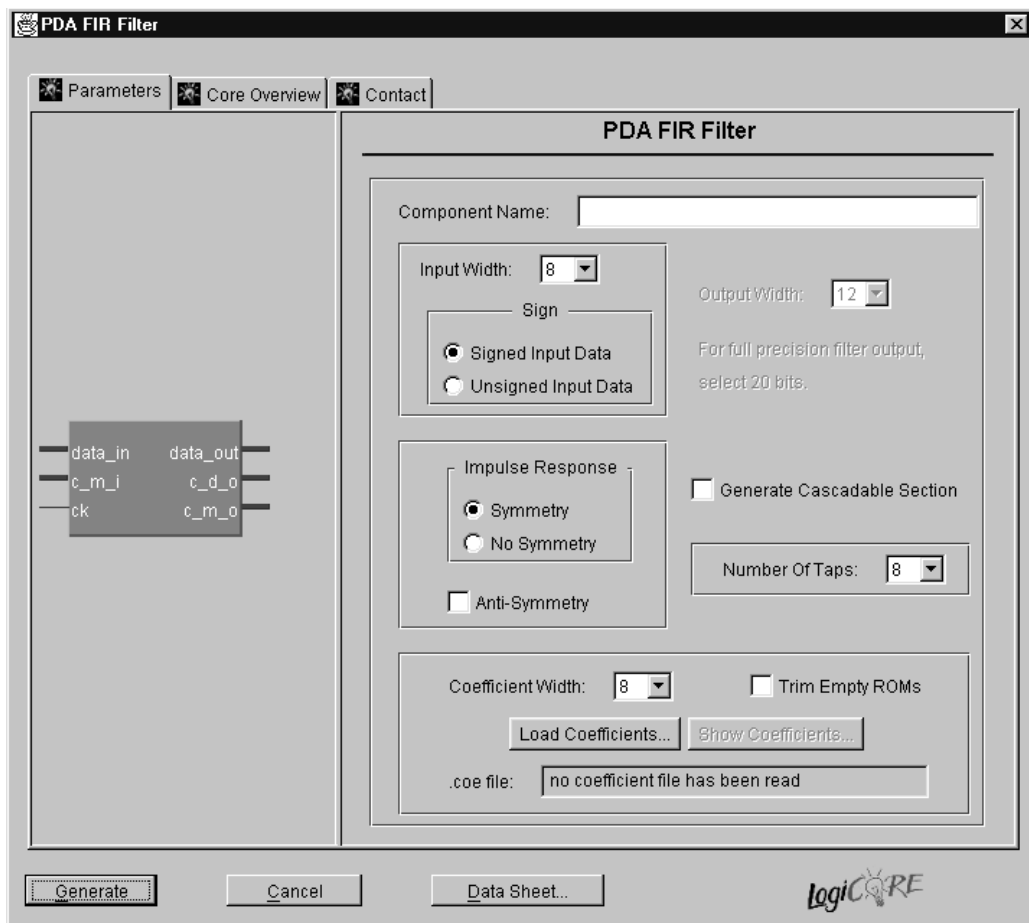


Figure 3-5 PDA FIR Filter Module Window

CoefData , **MemData** , **MEMORY_INITIALIZATION_VECTOR** , all must be the last keywords in the .COE file. Any keywords that follow them will be ignored.

Examples of .COE files for the PDA FIR, SDA FIR, distributed RAM, distributed ROM and Virtex block RAM COE files can be found in the \$XILINX/coregen/data directory and are reproduced below.

```
***** Example of PDA FIR .COE file with *****
***** hex coefficients - pdafir.coe *****
Component_Name=fltr16;
Number_Of_Taps=16;
Input_Width = 8;
Output_Width = 20;
Coefficient_Width = 12;
Impulse_Response_Symmetry = true;
Radix = 16;
CoefData=346,EDA,0D6,F91,079,FC8,053,FE2;
```

```
***** Example of PDA FIR .COE file with *****
***** decimal coefficients - pfir_dec.coe *****
Component_Name=fltr16;
Number_Of_Taps=16;
Input_Width = 8;
Signed_Input_Data = true;
Output_Width = 21;
Coefficient_Width = 8;
Impulse_Response_Symmetry = true;
Radix = 10;
CoefData=1,-3,7,9,78,80,127,-128;
```

```
***** Example of SDA FIR .COE file with *****
***** decimal coefficients - sdafir.coe *****
Component_Name=sdafir;
Number_Of_Taps=6;
Radix=10;
Input_Width=10;
Output_Width=24;
Coefficient_Width=11;
```

```
Impulse_Response_Symmetry = false;
CoefData= -1,18,122,418,-40,3;

***** Example of distributed RAM .COE file *****
***** with hex coefficients - ram_hex.coe *****
Component_Name=ram16x12;
Data_Width = 12;
Address_Width = 4;
Depth = 16;
Radix = 16;
memdata=346,EDA,0D6,F91,079,FC8,053,
FE2,03C,FF2,02D,FFB,022,002,01A,005;

***** Example of distributed ROM .COE file *****
***** with decimal coefficients - rom_dec.coe *****
Component_Name=rom32x8;
Data_Width = 8;
Address_Width = 5;
Depth = 32;
Radix = 10;
memdata=127,127,127,127,127,126,126,126,
125,125,125,4,3,2,0,-1,-2,-4,-5,-6,-8,-9,
-11,-12,-13,-38,-39,-41,-42,-44,-45,-128;

***** Example of Virtex single port *****
***** RAM .COE file with hex *****
***** coefficients - v_spbram.coe *****
Component_Name = v_spbram;
Depth = 256;
Data_Width = 32;
Radix = 16;
```

```
Default_Data = FFF;
MEMORY_INITIALIZATION_VECTOR =
FF0, F0F, 0FF, FF4, F4F, 4FF, FF8, F8F, 8FF;

***** Example of Virtex dual port *****
***** RAM .COE file with binary *****
***** coefficients - v_dpbram.coe *****
Component_Name=v_dpbram;
Depth_A = 4096;
Data_Width_A = 16;
Depth_B = 1024;
Data_Width_B = 64;
Radix = 2;
Default_Data = 10101010;
MEMORY_INITIALIZATION_VECTOR=
1111111111111110,
1111111111111101,
1111111111111011,
111111111110111;
```

Command Files

A Xilinx CORE Generator command file is a file that contains valid CORE Generator commands and comments (see the table below for the list of valid CORE Generator commands). Command file comment lines begin with a '#' symbol. The CORE Generator allows you to execute command files in GUI mode by selecting the File->Execute Command File... item in the main menu and entering the path to the command file, as well as in batch mode by invoking "coregen" in command line mode with the -b <command_file> command line option.

The four types of command files in the CORE Generator System-- .xco files, coregen.ini/coregen_<user_name>.ini files, user-generated command files, and coregen.log files, are described below.

coregen.ini/coregen_<user_name>.ini

.ini files are executed when CORE Generator is first invoked, and also when changing projects. .ini files should be used to set system properties such as AcrobatPath, or to set user customization properties such as CoreSelect. When CORE Generator System is invoked, it first searches the current working directory for a file named "coregen_<user_name>.ini", then for a file named "coregen.ini", and executes the first .ini file it finds. Alternatively, you can direct CORE Generator System to use a particular .ini file by specifying the

"-i <ini_file>"

command line option when invoking the CORE Generator System. .ini files are optional, and are not required for the CORE Generator System to operate correctly.

When changing projects, the CORE Generator System uses the same search order to find and execute an .ini file in the new project directory. Placing .ini files in project directories allows you to customize CORE Generator behavior for individual projects.

User-generated Command Files

Users can write their own command files to generate COREs, create projects, customize the CORE Generator environment, or execute any other CORE Generator command. User-generated command files may have any name and extension. All global property SET commands executed within a user-generated command file

are only in effect for that session. However, all project property SET commands executed within a user-generated command file will modify the current project.

See the tables included at the end of the *Command Files* section for the list of valid global and project properties that can be set within a command file.

.XCO Files

When a CORE is generated, a file called `<component_name>.xco` will be generated by the system. This file is a log file that records all the options used to create the CORE. It can be used to verify all the options that were used when the CORE was generated and can also be used to recreate the CORE exactly using the

File -> Execute Command File menu or in batch mode.

Comment lines begin with the # character. Any output format or system options lines start with the keyword SET. These options will match the options set in the `coregen_<User_Name>.ini`. The lines that start with CSET are the options that are passed from the CORE parameterization dialog window. All data read in from a .COE file is also listed preceded by the CSET keyword.

The following is an example for the FIR filter generated using the `pdfir.coe` file described above.

```
# Xilinx CORE Generator v2.1.11
# Username = roman
# FoundationPath = G:\Xilinx
# COREGenPath = d:\cg212\rtf\coregen
# ProjectPath = D:\Designs\cgvxtest
# ExpandedProjectPath = D:\Designs\cgvxtest
SET BusFormat = BusFormatParen
SET SimulationOutputProducts = VHDL
SET ViewlogicLibraryAlias = ""
SET XilinxFamily = XC4000
SET DesignFlow = VHDL
SET FlowVendor = Synplicity
SELECT PDA_FIR_Filter XC4000 Xilinx 1.0
```

```
CSET number_of_taps = 16
CSET component_name = fltr16
CSET trim_empty_roms = FALSE
CSET radix = 10
CSET impulse_response_symmetry = TRUE
CSET signed_input_data = TRUE
CSET generate_cascadable_section = FALSE
CSET coefdata = 1,-3,7,9,78,80,127,-128
CSET output_width = 15
CSET input_width = 8
CSET coefficient_width = 8
CSET antisymmetry = FALSE
GENERATE
```

coregen.log

Coregen.log is a log file that is automatically written to the current working directory by the CORE Generator System. corgen.log contains all the actions and messages performed during a CORE Generator session, so you can refer to it to see what occurred during that session. Since corgen.log is a command file it can also be replayed to recreate a CORE Generator session.

On PC's, corgen.log is written to **\$XILINX\coregen\tmp**

On Workstations, corgen.log is written to the current project directory.

Valid CORE Generator Commands

Command Syntax	Description
CSET <core_property> = <value>	Sets a core parameter value before generation.
END	Ends a CORE Generator session.
EXECUTE = <command_file_path>	Executes a CORE Generator command file.
GENERATE	Generates a core.
NEWPROJECT = <new_project_path>	Creates a new project.
SELECT <core_name> <architecture> <vendor> <core_version>	Selects the indicated core.
SET <global_property> = <value>	Sets a global property value.
SETPROJECT <project_path>	Makes the indicated project the active project.

Global Properties

Property Syntax	Description
AcrobatPath = <acrobat_install_directory>	Specify the path to the Acrobat install directory.
AcrobatName = <acrobat_exe_name>	Specify the Acrobat executable name.
CoreSelect = <SpecifiedVersion LatestVersion ProjectVersion>	Specify what version of a core to select.
FoundationPath = <path_to_foundation_tool>	Specify the path to the Foundation directory.
OverwriteFiles = <true false>	Specify whether Coregen will automatically overwrite design files during core generation.
ProjectOverride = <true false>	Specify whether the current project properties override the project properties in a command file.

Project Properties

Property Syntax	Description
BusFormat = <BusformatAngleBracket BusformatSquareBracket BusformatParen BusformatNoDelimiter>	Specify the desired bus delimiter (default = BusformatNoDelimiter)
DesignFlow = <Schematic VHDL Verilog>	Specify the desired design flow (default = schematic)
FlowVendor = <Foundation Viewlogic Mentor Cadence Synplicity Synopsys Exemplar Other>	Specify the desired CAE flow vendor (default = Viewlogic)
SimulationOutputProducts = <VHDL Verilog None>	Select the desired simulation products (default = None)
ViewlogicLibraryAlias = <library_alias_name>	Specify the viewlogic library alias (must match the library alias for the project library in your viewdraw.ini)
XilinxFamily = <XC4000 Spartan Virtex Spartan2>	Select the desired Xilinx architecture family (default = XC4000)

Generating COREs in Batch Mode

When the CORE Generator System is run with no options it comes up in GUI mode. The CORE Generator System can be run in batch mode to generate COREs by specifying the .XCO file that defines the CORE to be generated and its parameters, and the project directory where the output files should be deposited.

The .XCO file created by the CORE Generator System when run in GUI mode can be used to drive the generation of the same CORE in batch mode. These can be edited and renamed to generate a slightly different CORE. Note also that an .XCO file can contain the commands to generate more than one CORE.

If the directory where the CORE Generator executables resides is not in the command search path, the CORE Generator System must be invoked using a fully specified path.

The CORE Generator System's command line options are listed below.

- **-b** <command_file_name>

Tells the CORE Generator the name of the command file (suffix .XCO) that should be executed by the batch mode run.

- **-i <coregen_ini_file_name>**

By default the CORE Generator System will use the profile that is in the specified project directory. If a different profile is required to be used then it can be explicitly specified using a fully specified path name.

- **-p <project_path>**

Specifies the project directory. The project path must be fully specified.

- **-q <polling_dir_path>**

This is an option for third party tools that call the CORE Generator System and should not be used by users in batch mode.

- **-h**

Displays the CORE Generator batch mode help screen.

```
D:\Designs\filter> coregen -h
```

```
Xilinx CORE Generator v2.1i
```

Usage:

<command_file_name> is the path to the Command file to be executed.

<coregen_ini_file_name> is the path to the Coregen ini file to be loaded, or it will load this file from the Project directory.

<project_path> is the path to the Coregen Project.

<polling_dir_path> is the polling directory.

-h displays this usage message.

Typical invocation of the CORE Generator System in batch mode would be as follows:

```
coregen -b <core_name>.xco -p <project_path>
```

Updating COREs in the CORE Generator System

Version Handling in the CORE Generator System

The CORE Generator System is capable of handling multiple versions of a CORE. The ability to create newer versions of a CORE allows a designer to introduce additional functionality to an existing CORE and fix any problems that may have been found in an earlier version of the CORE.

When a new project is created the COREs displayed in the CORE Generator System's main window are the latest versions of the COREs.

When new COREs and new versions of existing COREs are downloaded from the IP Center they are installed in the CORE Generator System's hierarchy, but are not visible for existing projects. This capability exists to insulate existing projects from updates to the COREs used in that project and the changes in the functionality that might occur if this were performed automatically. The capability exists to allow a new CORE or a new version of an existing CORE to be made available in an existing project, should this be required.

Downloading New COREs

The new COREs and their installation instructions can be downloaded from the Xilinx IP Center at <http://www.xilinx.com/>

Links to the IP Center are available from the menus via

Core -> CoreLINX or Web -> CoreLINX

A toolbar button is also available to access the IP Center.

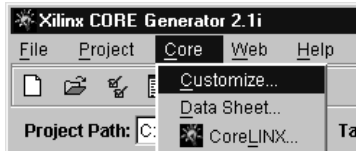
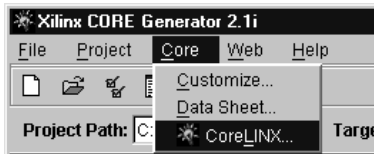


Figure 3-6 CoreLINX Buttons



Figure 3-7 Browse CoreLINX Button

CORE Version Update in an Existing Project

As described above, an existing project does not automatically adopt new COREs or new versions of existing COREs. If new COREs are downloaded and it is required that they be visible in an existing project there are two ways to do this.

In general, make sure when updating an existing project to use new versions of COREs because there may be differences that may change the design operation. The data sheet for the new version of the CORE should be read carefully to ensure that the differences are understood within the context of the design in progress and are not likely to affect adversely the functionality of the design.

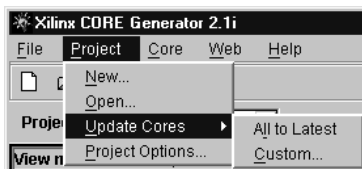


Figure 3-8 Update COREs Button

The **Project -> Update Cores -> All to Latest**, menu selection will update the project files for the currently selected project to reflect the latest version of all COREs. Care should be taken when using this option because of the global nature of the change for the current project.

Individual COREs can be added or removed to the current project's profile using the **Project -> Update Cores -> Custom** menu selection. This displays the Update Project COREs dialogue screen which allows the user to go through the listings of the COREs in each category and click on the check box adjacent to any new COREs that

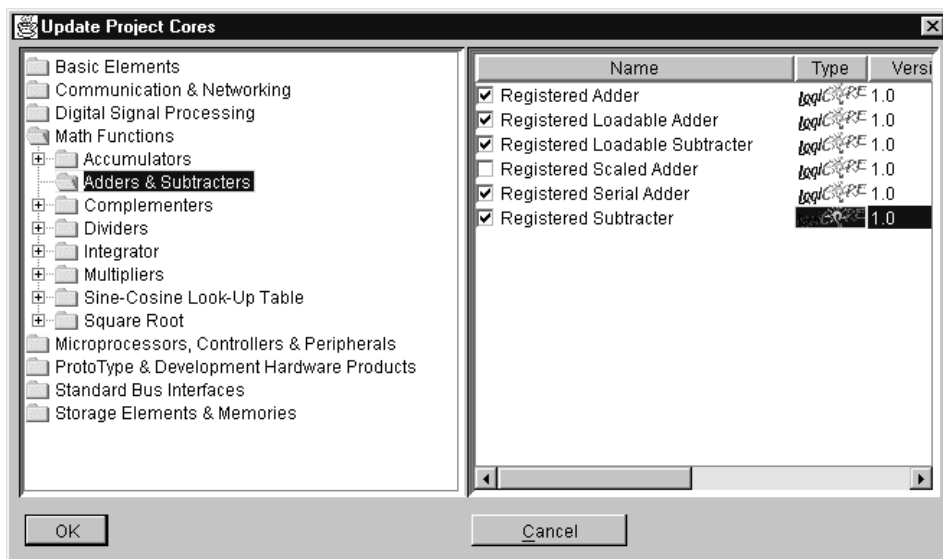


Figure 3-9 Update COREs Custom Window

they want to be available in the current project. By clicking on a check box that is already selected a CORE that is currently available can be removed.

CORE Generator Design Flows

Overview

This chapter describes how to integrate a CORE Generator module into a user design. The chapter covers the two main types of design flows: schematic design flow and the HDL design flow.

The CORE Generator design environment currently supports the following schematic design tools:

- Viewlogic
- Foundation
- Mentor
- Cadence*

The CORE Generator design environment currently supports the following HDL Synthesis tools:

- Synopsys FPGA Express
- Synopsys FPGA Compiler
- Exemplar
- Synplicity

*Limited Cadence support is currently available as described in the *Schematic Design Flow* Section.

CORE Generator Design Flow Basics

The CORE Generator System produces an EDIF Netlist, and may also produce a Foundation or Viewlogic schematic symbol, and /or a VEO or VHO template file. The EDN file contains the information for implementing the module. The Foundation or Viewlogic symbol allows you to integrate the CORE Generator module into a schematic for those CAE tools. Finally, the VEO and VHO template files contain code that can be used as a model to instantiate the module in a Verilog or VHDL design so that it can be simulated and integrated into a design as a black box.

The dotted circles in Figure 4-1 highlight the portions of the flow directly associated with the CORE Generator Design Flow. The circle on the left shows the EDN, VEO, VHO and schematic files produced by the CORE Generator System as discussed above. The circle on the right shows the `XilinxCoreLib` and `<Vendor>CoreLib` libraries that are extracted by the `get_models` utility. These libraries contain the behavioral simulation models for the CORE Generator cores.

Please refer to the *Design Flows Chapter* for more information on the various CAE flows depicted in this diagram as well as for information on the `get_models` utility.

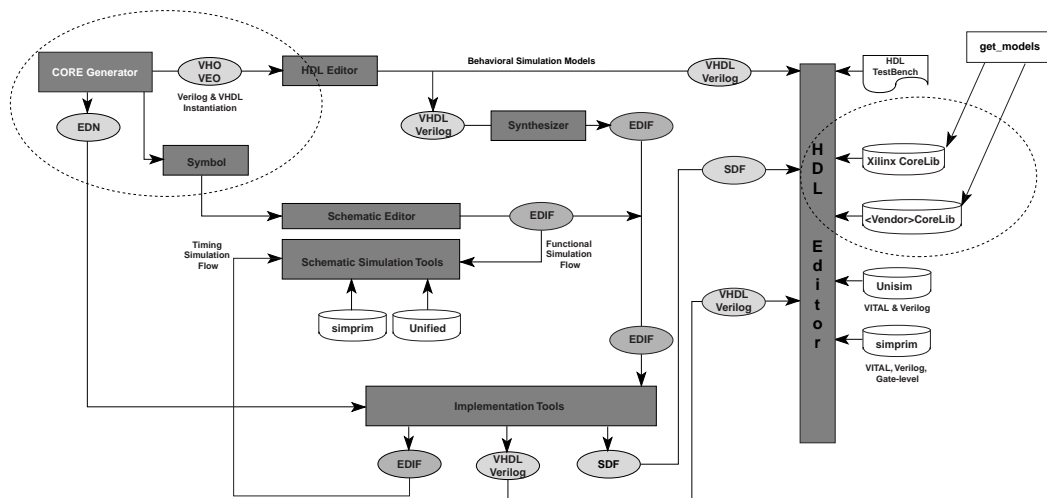


Figure 4-1 CORE Generator Design Flows

Viewlogic Design Flow

The Viewlogic interface outputs are generated by a script, `vlink`, which calls for both Viewlogic tools and Xilinx Implementation Tools programs.

On a Windows platform you must have both the Viewlogic and Xilinx Implementation Tools software installed to generate the outputs required to integrate a core into a Viewlogic design.

On a UNIX platform, your environment must be set up to run the Viewlogic tools and the Xilinx Implementation software, as well as the CORE Generator System software. If any of these tools are either not installed or not properly set up in your environment, diagnostic errors may be reported in the `vlink.log` file written to your project directory.

Note: For details on Viewlogic and Xilinx Implementation Tools setup, please refer to the *Alliance Quick Start User Guide*. For both platforms, the CORE Generator project must be located in a valid Viewlogic project directory.

The Viewlogic design flow is described below.

Create a directory for a Viewlogic project

Example

```
c:\wvoffice\project
```

Set up project libraries

On Workstations, these libraries must be defined in the `viewdraw.ini` file located in the project's working directory.

On PCs, these libraries must be set up through the Project Manager GUI as described below:

- Open a New project.
- Add a configured FPGA library from the list. As an example, if the `xc4000XL` is selected, the `xc4000x`, `logiblox`, `simprims`, `builtin` and `xbuiltin` libraries are added automatically.
- Add the project directory as a Writeable library and give it the alias, "primary". Move it to the top of the library search order.

- Save the project in the Viewlogic Project Manager.

The following is an example of the library search order needed to create an XC4000XL design:

```
dir [p] c:\wvoffice\project (primary)
dir [rm] %XILINX%\viewlog\data\xc4000x (xc4000x)
dir [r] %XILINX%\viewlog\data\logiblox (logiblox)
dir [rm] %XILINX%\viewlog\data\simprims (simprims)
dir [rm] %XILINX%\viewlog\data\builtin (builtin)
dir [rm] %XILINX%\viewlog\data\xbuiltin (xbuiltin)
```

Note: The (primary) alias is very important since the CORE Generator System will look for it in order to define the directory to copy the symbol and simulation files. This alias should match the one specified in the CORE Generator System Options under Viewlogic Library alias.

Set the Output Format

From the CORE Generator Project menu, select Project Options and check the following options:

- Design Flow: Schematic
- Vendor: Viewlogic

The netlist bus format is automatically set to < BI > when Viewlogic is specified as the vendor.

Set Project Path and Viewlogic Library Alias

From the CORE Generator Project menu, select Project / Open and ensure that the Project Path is set to point to the Viewlogic project directory you are working on (c:\wvoffice\project). If the desired project is not on your list of displayed projects, use the **Browse** button to navigate to this directory. Make sure that the string you enter in the CORE Generator window for the Viewlogic Library Alias matches the one defined in the viewdraw.ini file. The default Viewlogic Library Alias is “primary” but any name up to 8 characters can be used.

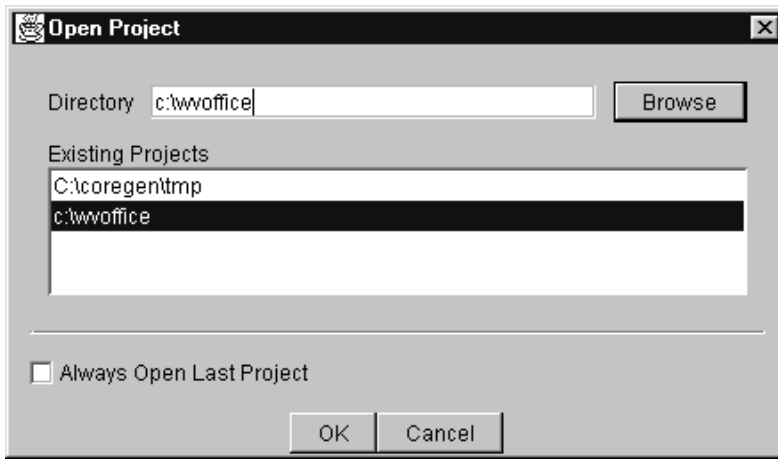


Figure 4-2 Project Options GUI

Select Desired Module

Select the module you want to generate by navigating around the module browser and clicking on the desired module. You may click the SPEC button on the CORE Generator toolbar to review the module's datasheet.

Double click on the selected module to reveal its parameterization window. When you have entered all the parameterization details required by the module, click the Generate button.

Output Files

A Viewlogic Symbol, a Viewlogic simulation file (WIR) and a Netlist File (.EDN) will be created.

Note: The symbol is created with a Block type of "Composite" and placed in the SYM subdirectory within the Viewlogic project. This symbol will not necessarily match the CORE Generator module symbol shown in the datasheets in shape and pin order. It is possible to manually modify your symbol using the Viewlogic Symbol Editor as needed.

- The simulation file is created from the EDN file and placed in the WIR subdirectory within the Viewlogic project.

- The Netlist (.EDN) used for implementation, is placed directly in the Viewlogic project directory.

Note: The WIR file is used by Viewlogic to perform Functional simulation and should not be deleted. In order to generate this file the CORE Generator System needs to access some Xilinx Implementation Tools executables and may fail if this tool is not set up properly.

Note: Workview Office 7.4 and 7.5 and Powerview 6.0 do not support Virtex and Spartan2 simulation, so only a symbol and a netlist are generated for these cores. You must use either VHDL or Verilog to simulate Virtex designs in Viewlogic.

If an error occurs during the generation of these files, check the vlink.log file located in the Project Directory.

Load Symbol in Schematic Editor

Open the Viewlogic schematic tool, load your top level schematic (or create a new one) and add the new symbol for the module you have just created. Finally, attach a “LEVEL” property to the symbol with a value of “XILINX”. Check that this property has been added properly to the symbol by displaying all properties attached to the symbol.

From this point onward, the flow for processing this design is the same as if you were using macros from the Unified Library. For further information, refer to the *Viewlogic Interface/Tutorial Guide*.

Note: When executing the Viewlogic “Check” program, error messages like the one shown below will be displayed for every CORE Generator module but can be safely ignored.

Example

```
ERROR: Could not load schematic sheet: corename.1
```

Foundation Design Flow

In the 2.1i release of the CORE Generator System is integrated into the Foundation Schematic Editor. Please refer to the F2.1i documentation for details on integrating your CORE Generator module into a Foundation Schematic Design.

Mentor Design Flow

The 2.1i release of the CORE Generator System is integrated into the Mentor Design Architect. Please refer to the *Mentor Interface* documentation for details on integrating your CORE Generator module into a Mentor Schematic Design.

Cadence Design Flow

Setting the Vendor to Cadence in the CORE Generator Project Options dialog window will direct the application to generate and EDIF Implementation netlist with the proper bus delimiter format for Concept-HDL.

For further information integrating a core into a Concept-HDL schematic, please refer to *Solution 2005* at:

<http://www.support.xilinx.com>.

The CORE Generator System will be integrated into Concept-HDL in a future release.

HDL Design Flows

This section describes the elements of an HDL design flow in the CORE Generator System environment.

The first part of this section covers the new behavioral model delivery system in the CORE Generator System including the new `get_models` utility and the new `.VEO` and `.VHO` Verilog and VHDL template files. This part is followed by the specifics of the vendor flows for Synopsys FPGA Express, Synopsys FPGA Compiler, Exemplar, and Synplicity tools in combination with the Verilog-XL and MTI VHDL and Verilog simulators.

The following are the basic steps involved in integrating a CORE Generator module into an HDL design:

- Generate the module
- Instantiate the module in your design
- Perform a behavioral simulation of your design with the integrated module
 - Extract the CORE Generator module behavioral models from the CORE Generator tree with `get_models`

- Analyze the models
- Simulate
- Implement the design.

Behavioral Model Delivery in the CORE Generator System

In the 2.1i release of the Xilinx CORE Generator, the HDL behavioral models have been rewritten to support “optional pins” on CORE Generator modules. This refers to the fact that the various input and output ports on a CORE Generator module may not all be required by the user. Some pins (the “optional” ones) may be omitted when the module is generated. With the new 2.1i CORE Generator parameterized behavioral models and internal representations, when optional ports are not specified for the module, the construction of, and behavioral models for, the module are adjusted accordingly. The unused pins and their associated logic are omitted when the CORE Generator System generates the EDIF implementation netlist for the module. Since much of the unnecessary logic is omitted up front from the implementation netlist generated by the CORE Generator, there is less reliance on the Xilinx Mapper to trim out extraneous logic.

Several other aspects of CORE Generator HDL behavioral model delivery have also changed for customers using CORE Generator modules in HDL design flows. The additions are listed as follows:

- A new utility called `get_models` has been added. Its purpose is to extract the HDL behavioral models from the CORE Generator tree into a library directory.
- Behavioral models for each CORE Generator module are no longer copied to the user’s project directory. Instead, CORE Generator System 2.1i only writes out HDL instantiation template files (`.VEO` and `.VHO`) which contain pointers to the generic, parameterized HDL simulation models in a `get_models` extracted library or directory.
- In those cases where a behavioral model is hierarchical, pointers to the lower level behavioral models referenced by a higher level one are also written to the instantiation template files.

get_models

This section describes the new methodology you will need to follow to incorporate Xilinx CORE Generator HDL behavioral models into your flow.

The `get_models` program is a command line utility used to extract the Verilog or VHDL behavioral models embedded within a user's CORE Generator System installation to a single, central location. The models may exist in archived or source file format. In the case of an HDL flow using a compiled simulator, extracting the models to a single directory allows them to be conveniently analyzed for a particular Verilog or VHDL simulator. In a Verilog flow using an interpretive simulator, `get_models` simply extracts and collects together the Verilog behavioral models of all installed CORE Generator modules so that they can be conveniently referenced during simulation.

Alternatively, `get_models` can also be run to extract individual behavioral models for specific CORE Generator modules to a user's project directory, if preferred.

Note: Since the behavioral model for any new CORE Generator module is always embedded into that module's directory archive, you must also run `get_models` whenever you install a CORE Generator IP module update.

Syntax

```
get_models [ -verilog | -vhdl ] <destination_directory>
```

Required Parameters

- `-verilog` | `-vhdl`: Extract Verilog (or VHDL) behavioral models only. You must specify either `-verilog` or `-vhdl` when running `get_models`.
- `<destination_directory>`: name of directory in which the various `<Vendor>CoreLib` subdirectories are to be created. Typically a `XilinxCoreLib` subdirectory will always be one of the directories created in this directory.

Recommended location of `<destination_directory>` is `$XILINX/data`. (On networked UNIX workstations, system administrator privileges may be required to use this location if multiple users are using the same installation).

Note: You must specify either `-verilog` or `-vhdl` as one of the arguments to `get_models`.

Inputs

The inputs to the `get_models` utility are the CORE Generator behavioral models located in your Xilinx CORE Generator System installation under `<install_directory>/ip/xilinx`. These models may exist in either of two formats:

- Archived together with other data files related to a given module as a .JAR (JAVA Archive format) file, located under `$XILINX/coregen/ip/com/xilinx`.
- Non-archived, source file format (.v and .vhd files) located in the respective CORE Generator module directory, under `$XILINX/coregen/ip/com/xilinx`.

Outputs

The outputs to the `get_models` utility is a directory of extracted source format Verilog or VHDL behavioral models as listed in the table below.

Output Files	Description
*.v	Verilog behavioral models extracted from the CORE Generator directory tree
*.vhd	VHDL behavioral models extracted from the CORE Generator tree
<module>_comp.vhd	VHDL component declaration files for each CORE Generator module extracted from the CORE Generator tree

.VEO (Verilog Instantiation Template)

A .VEO (Verilog Output) file is a file containing code that can be used to instantiate your CORE Generator module into your Verilog design, and also contains code that supports behavioral simulation.

A sample Verilog instantiation template for an 8-bit adder is shown below. The main components are:

- Simulation Model “include” statement
- Instantiation code
- Module Port Declaration with customization parameters

```
/*The following line MUST appear at the top of the file in
which the instantiation will be made:*/
// LIB_TAG
`include "XilinxCoreLib/adreVHT.v"
// LIB_TAG_END

/*The following is an example of an instantiation. Cut and
paste this code into your design, changing the instance name
and port connections (in parentheses) to your own signal
names.*/
// INST_TAG
adder8 YourInstanceName (
    .A(A),
    .B(B),
    .C(C),
    .CE(CE),
    .CI(CI),
    .CLR(CLR),
    .S(S));
// INST_TAG_END

*/Cut and paste this code into your design, after the module
in which it is to be instantiated.*/
// MOD_TAG
module adder8 (
    A,
    B,
    C,
    CE,
    CI,
    CLR,
    S);
input [7 : 0] A;
input [7 : 0] B;
input C;
input CE;
```

```
input CI;
input CLR;
output [8 : 0] S;
    ADREVHT #(8,
    1)
    inst (.A(A),
    .B(B),
    .C(C),
    .CE(CE),
    .CI(CI),
    .CLR(CLR),
    .S(S));
endmodule
// MOD_TAG_END
```

.VHO (VHDL Instantiation Template)

The .VHO instantiation template file is a file containing code that can be used to instantiate a CORE Generator module into a VHDL design, and also contains code that supports behavioral simulation. The .VHO file is similar to the .VHI instantiation template generated by the 1.4 and 1.5 versions of the CORE Generator, but is distinguished by an additional VHDL CONFIGURATION snippet that must be added to a CONFIGURATION declaration in your VHDL testbench file or upper level design file. The Configuration sets the values of various VHDL generics used to customize the CORE Generator VHDL simulation models in the 2.1i release.

The main parts of a .VHO file are:

- XilinxCoreLib LIBRARY Declaration
- COMPONENT Declaration
- Instantiation template
- CONFIGURATION Declaration with VHDL generics

Sample .VHO file for an 8-bit adder

```
--User: Make sure that these statements appear
--above the top-level entity declaration in your VHDL
design...
--LIB_TAG
Library XilinxCoreLib;
```

```
Use XilinxCoreLib.null_comp.all;
-- LIB_TAG_END
-- User: Make sure that this statement appears
-- in the architecture header in your VHDL design...
-- COMP_TAG
component adder8
  port (
    a: IN std_logic_VECTOR(7 downto 0);
    b: IN std_logic_VECTOR(7 downto 0);
    c: IN std_logic;
    ce: IN std_logic;
    ci: IN std_logic;
    clr: IN std_logic;
    s: OUT std_logic_VECTOR(8 downto 0));
end component;
-- COMP_TAG_END
-- User: Make sure that this statement appears
-- in the architecture body in your VHDL design,
-- substituting your own instance name where shown.
-- Do not forget to change the net names in the port map
-- to your own design's net names.
-- INST_TAG
your_instance_name : adder8
  port map (
    a => a,
    b => b,
    c => c,
    ce => ce,
    ci => ci,
    clr => clr,
    s => s);
-- INST_TAG_END
-- User: Make sure that this text appears
-- within the top-level configuration body in your VHDL
design,
-- for example:
--
-- configuration cfg_top of top_level is
--   for arch_name
--     <Insert text here>
```

```
--      end for;
-- end cfg_top;
--
-- CONF_TAG
for all : adder8 use entity XilinxCoreLib.null(behavioral)
  generic map(
    signed => true,
    input_width => 8);
  end for;
CONF_TAG_END
```

Note: The lines between the two markers, -- CONF_TAG, and - CONF_TAG_END:

-CONF_TAG

```
for all : adder8 use entity XilinxCoreLib.null(behavioral)
  generic map(
    signed => true,
    input_width => 8);
  end for;
-- CONF_TAG_END
```

These lines must be added to a CONFIGURATION statement in your VHDL test fixture file or top level design file.

CORE Generator Verilog Flow

This section describes the procedure for behavioral simulation, synthesis, and implementation of Verilog designs containing CORE Generator modules. Specific instructions are included for the following tools:

Synthesis

- Synopsys FPGA Compiler
- Synopsys FPGA Express
- Synplicity Synplify
- Exemplar Leonardo

Simulation

- MTI ModelSim/VLOG
- Cadence Verilog-XL

Basic Steps in the Verilog Design Flow

Module Generation

Specify the Design Entry, Vendor, and Behavioral Simulation settings for the project.

Behavioral Simulation

Prepare for simulation

1. Extract the behavioral models to a (source) library
2. Instantiate the module in the parent design
 - a) Edit the include statement to reflect the actual location of the extracted behavioral model source library
 - b) Comment out the instantiation template in the .VEO file.
 - c) Copy the .VEO file to <module_name>.v
 - d) Edit the connections in the module declaration
3. Create a testbench

- a) Perform the behavioral simulation
4. Synthesis
 - a) a. Apply any “black-box” properties, if required
 - b) b. Synthesize the design
 - c) c. Write out the implementation netlist
5. Implementation

Implement the design using the Xilinx tools

Module Generation

Specify the Design Entry, Vendor, and Behavioral Simulation settings for the project.

From the CORE Generator Project menu, select Project Options and specify the Design Flow as “Verilog”, and specify the synthesis vendor software you will be using to synthesize your VERILOG design. This will specify the appropriate EDIF bus delimiter format required for that flow for proper integration with the upper level “parent” implementation netlist written out by the synthesizer for your design.

In the example below, the Design Flow has been set to “Verilog”, and the Vendor has been set to “Synopsys”, and as a result, the bus delimiter format is automatically set to “B<I>”.

Behavioral Simulation

Preparing for Verilog Behavioral Simulation

Note: The following instructions assume that the CORE Generator System v2.1i has been installed in the directory \$XILINX/coregen.

Extract the CORE Generator Verilog Behavioral Models to a (source) library (required for all simulators).

Prior to simulating any CORE Generator cores, the behavioral models for the cores must be extracted from the CORE Generator system installation area to a directory of the user's choosing (referred to here as “<destination_directory>”). The recommended location of “<destination_directory>” is \$XILINX/verilog/src, as this is the location of the source code for all other Xilinx Verilog libraries.

The `get_models` utility must be used to extract these models to the `<destination_directory>`. For a Verilog design flow, the `get_models` utility should be run with the `-verilog` switch to specify that only the Verilog models should be extracted. The full syntax of the command is:

```
get_models -verilog <destination_directory>
```

Within the specified `<destination_directory>`, `get_models` creates a subdirectory for each vendor whose cores exist in your CORE Generator installation. The names of these directories take the following form:

```
<Vendor>CoreLib
```

Note: The first letter of the vendor name is capitalized.

As an example, the subdirectory `get_models` creates for the Xilinx IP models is `XilinxCoreLib`.

`Get_models` extracts all Verilog models provided by a particular vendor from the CORE Generator installation area to the appropriate `<Vendor>CoreLib` subdirectory when it is invoked with the `-verilog` option.

(For more information on `get_models`, please refer to the `get_models` section of the *CORE Generator Design Flows* chapter).

Instantiate the core (same procedure for all simulators and synthesis tools)

Selecting the Verilog checkbox under the Behavioral Simulation option in the CORE Generator Project Options window directs CORE Generator System to generate a Verilog template file (`<component_name>.VEO`) in addition to an implementation netlist (`<component_name>.EDN`) whenever a core is generated. The `.VEO` template file includes:

- a library `"`include"` statement
- a module declaration section
- a module instantiation snippet

These elements should be copied and pasted into the parent design, and the following modifications made:

Edit the include statement in the .VEO file to reflect the actual location of the extracted behavioral model source library.

Cadence Verilog-XL and MTI ModelSim/VLOG

The library “`include” statement(s) must be modified to reflect the actual location of the XilinxCoreLib and any other <Vendor>CoreLib libraries extracted by get_models.

Example

```
`include /tools/xilinx/verilog/src/XilinxCoreLib
```

Comment out the instantiation template in the .VEO file.

Copy the .VEO file to <module_name>.v

Connect the core to the parent design by editing the module connections.

The dummy signals in the CORE Generator module instantiation snippet must be replaced with the actual signals in the parent design to which the component is to be connected.

The module declaration and component instantiation establish a link in the parent Verilog design to the EDIF implementation netlist for the CORE Generator module. This link is necessary to ensure that the design will be implemented properly after the parent Verilog design has been synthesized. The core's EDIF netlist is merged in with the rest of the parent design by the Xilinx NGDBUILD tool during the translation phase of the design flow.

Example

The following example illustrates the use of the Verilog template file with a parent design. In this example, an 8-bit adder core, myadder8, is generated using the CORE Generator system and is instantiated in a parent design called "myadder8_top". "myadder8" represents an 8-bit adder. The relevant CORE Generator files in this example are the instantiation template file, myadder8.veo, and the parent design, myadder8_top.v which are list below.

VERILOG instantiation template file: myadder8.veo

```

/*****
This file was created by the Xilinx CORE Generator tool,
and is (c) Xilinx, Inc. 1998, 1999. No part of this file may
be transmitted to any third party (other than intended by
Xilinx) or used without a Xilinx programmable or hardware
device without Xilinx's prior written permission.
*****/

// The following line must appear at the top of the file /
/ in which the core instantiation will be made.
//Ensure that the translate_off/_on compiler directives
//are correct for your synthesis tool(s)
/-- Begin Cut here for LIBRARY inclusion ---// LIB_TAG
// synopsys translate_off
`include "XilinxCoreLib/adreVHT.v"
// synopsys translate_on
// LIB_TAG_END ----- End LIBRARY inclusion -----
// The following code must appear after the module in
//which it is to be instantiated. Ensure that the
translate_off/_on compiler directives are correct for
//your synthesis tool(s).

/-- Begin Cut here for MODULE Declaration --// MOD_TAG
module myadder8 (
A,
B,
C,
CE,
CI,
CLR,
S);

input [7 : 0] A;
input [7 : 0] B;
input C;
input CE;
input CI;
input CLR;
output [8 : 0] S;

```

```
// synopsys translate_off

ADREVHT #(
8,
1)
inst (
.A(A),
.B(B),
.C(C),
.CE(CE),
.CI(CI),
.CLR(CLR),
.S(S));

// synopsys translate_on
endmodule

// MOD_TAG_END ----- End MODULE Declaration -----
// The following must be inserted into your Verilog file //for
this core to be instantiated. Change the instance
//name and port connections(in parentheses) to your own
//signal names.

//--Begin Cut here for INSTANTIATION Template--// INST_TAG
myadder8 YourInstanceName (
.A(A),
.B(B),
.C(C),
.CE(CE),
.CI(CI),
.CLR(CLR),
.S(S));

// INST_TAG_END ----- End INSTANTIATION Template -----
```

The .VEO file can be copied to "myadder8.v" and used to reference the behavioral model for the adder after commenting out the Instantiation Template section of the file using "/* */" comment markers as follows:

```
/* myadder8 YourInstanceName (
.A(A),
```

```

        .B(B) ,
        .C(C) ,
        .CE(CE) ,
        .CI(CI) ,
        .CLR(CLR) ,
        .S(S) ;
*/

```

The resulting `myadder8.v` file should be analyzed along with the parent design when preparing for simulation.

Now take a look at the parent design, `myadder8_top.v`. Notice that the ``include` statement, the module declaration and the instantiation (with dummy signal names replaced with actual signal names) have been copied and pasted from `myadder8.veo`.

VERILOG parent design file: `myadder8_top.v`

The component, `myadder8`, is instantiated and the module is declared. The module declaration and instantiation template are copied from `myadder8.veo`, and are copied and pasted into the parent design.

The user-specified instance name of `myadder8_1` replaces "YourInstanceName", and dummy signal names are replaced with actual signal names. Sections of code beginning with `// synopsys translate_off` and ending with `//synopsys translate_on` directives are ignored by the synthesizer and are used for simulation only.

Note: This directive is supported by Synopsys FPGA Compiler, Foundation Express, FPGA Express, Exemplar and Synplicity synthesis tools.

```

//-----
// synopsys translate_off
// edit the next line to reflect the actual path to
XilinxCoreLib
`include "/tools/xilinx/data/verilog/src/XilinxCoreLib/
adreVHT.v"
// synopsys translate_on
module top (A_P, B_P, C_P, CE_P, CI_P, CLR_P, S_P);
input [7 : 0] A_P;

```

```
input [7 : 0] B_P;
input C_P;
input CE_P;
input CI_P;
input CLR_P;
output [8 : 0] S_P;
// INST_TAG
myadder8 #(8, 1) myadder8_1 (
    .A(A_P),
    .B(B_P),
    .C(C_P),
    .CE(CE_P),
    .CI(CI_P),
    .CLR(CLR_P),
    .S(S_P));
// INST_TAG_END

endmodule
module myadder8 (
    A,
    B,
    C,
    CE,
    CI,
    CLR,
    S);
input [7 : 0] A;
input [7 : 0] B;
input C;
input CE;
input CI;
input CLR;
output [8 : 0] S;
// synopsys translate_off
ADREVTHT #(8, 1) inst (.A(A_P),
    .B(B_P),
    .C(C_P),
    .CE(CE_P),
    .CI(CI_P),
    .CLR(CLR_P),
```

```

        .S(S_P));
// synopsys translate_on
endmodule

```

Create The Testbench

To simulate a parent design that contains the myadder8 core, a testbench file, testbench.v, must be written that includes an instantiation of the parent design. The testbench must include stimulus to activate the adder. The framework for a testbench used to simulate this design is shown below with some sample simulation stimulus.

```

`timescale 1 ns/1 ps
module testbench;
    reg C;
    reg CE;
    reg CI;
    reg CLR;
    reg [7:0] A;
    reg [7:0] B;
    wire [8:0] S;
/* Instantiation of top level design */
    top uut (
        .C_P (C),
        .CE_P (CE),
        .CI_P (CI),
        .CLR_P (CLR),
        .A_P (A),
        .B_P (B),
        .S_P (S)
    );
/* Add stimulus here */
    always #10 C = ~C;
    initial begin
        $timeformat(-9,3,"ns",12);
    end
    initial begin
        CI = 0;
        A = 0;
        B = 0;

```

```
        CE = 1;
        C = 1;
        CLR = 1;
#100
CLR=0;
#20;
A = 8'b10000000;
B = 8'b00000001;
#40;
A= 8'b11100001;
#40
B= 8'b00000010;
#1000 $stop;
// #1000 $finish;
    end
/* end stimulus section */
endmodule
```

Perform the Behavioral Simulation

Certain vendors require that Verilog simulation netlists be analyzed before simulation can proceed. For example, if you are using Model Technology's ModelSim simulation tool to simulate your design, both the parent netlist and the testbench must be analyzed. The simulation files may be analyzed (using the vlog command) into a local, default, work library called "work", which is created using the vlib command.

MTI ModelSIM

Execute the following commands in the project directory:

```
vlib work
vlog testbench.v
vlog myadder8_top.v
vlog myadder8.v
```

The simulator may now be invoked:

```
vsim <top_level_module>
```

This will load the testbench, the parent design and the simulation model of the 8-bit adder core, stored in the subdirectory, XilinxCoreLib.

Cadence Verilog-XL

```
verilog testbench.v myadder8_top.v myadder8.v
```

Synthesis

The parent design containing the core or cores must next be synthesized. To do this, the synthesizer must be directed to treat each core as a "black-box" because the logic for each core is specified only in its EDIF implementation netlist <component_name.EDN>, not in any Verilog file.

Synopsys FPGA Compiler

Apply the "don't_touch" attribute to the module via the Synopsys compile script.

Synopsys FPGA Express

Make sure that you do NOT read in a separate .V or EDIF file for the CORE Generator module. FPGA Express will automatically treat the module as a black box.

Synplicity Synplify

Apply the "/* synthesis black_box */" attribute to the component instantiation as follows (this step is optional, but will prevent "black-box" warnings from Synplify during compilation):

```
/ Verilog black box example
// synopsys translate_on
module top (A_P, B_P, C_P, CE_P, CI_P, CLR_P, S_P);
input [7 : 0] A_P;
input [7 : 0] B_P;
input C_P;
input CE_P;
```

```
input CI_P;
input CLR_P;
output [8 : 0] S_P;

// INST_TAG

myadder8 #(8, 1) myadder8_1 (
    .A(A_P),
    .B(B_P),
    .C(C_P),
    .CE(CE_P),
    .CI(CI_P),
    .CLR(CLR_P),
    .S(S_P)) /* synthesis black_box */;
// INST_TAG_END
endmodule
```

Exemplar Leonardo

Make sure that you do NOT read in a separate .V or EDIF file for the CORE Generator module. Exemplar will automatically treat the module as a black box.

Write out the implementation netlist for the synthesized design

After the parent design has been synthesized, you must write out its implementation netlist using the synthesis tool. Depending on the synthesis tool being used and the target architecture, this implementation netlist may be an EDIF or XNF file, or a set of EDIF or XNF files.

Note: CORE Generator System breaks buses out into their component bits when writing out the EDIF implementation netlist for a module. This formerly created pin mismatch problems with the upper level EDIF written out by some synthesis tools. However, beginning with the 1.5 release of the Xilinx Implementation tools, EDIF2NGD will automatically resolve connections between bus nets written out in bus format (for example, `address<7:0>`) in a parent EDIF netlist, and bus nets written out as individual bits in a lower level EDIF.

Synopsys FPGA Compiler

No special instructions. FPGA Compiler will write out an SEDIF or SXNF file for 4K designs, and SEDIF for Virtex designs.

Synopsys FPGA Express

No special instructions. When you direct FPGA Express to “Export the Netlist” Express will write out an XNF file for 4K designs, and EDIF for Virtex designs.

Synplicity Synplify

Synplify can write out either XNF or EDIF netlists for both XC4000 designs. It will write out only EDIF for Virtex designs.

Exemplar

Write out the implementation netlist in EDIF format.

Implementation

The implementation netlists for each of the cores in the parent design are merged in with the main design when the ngdbuild program (the “Translate” stage of the Xilinx Design Manager) is run on the top level parent design during design implementation. To merge the netlists successfully, make sure that all of the CORE Generator .EDN EDIF netlist(s) for the generated module or modules are located in the same directory as the top level EDIF netlist for the synthesized design. Alternatively, you can run NGDBUILD with the `-sd` option, which will allow you to specify explicitly the location of the directory containing the CORE Generator EDN files.

CORE Generator VHDL Flow

This section describes the procedure for behavioral simulation, synthesis, and implementation of VHDL designs containing CORE Generator modules. Specific instructions are included for the following tools:

Synthesis

- Synopsys FPGA Compiler
- Synopsys FPGA Express
- Synplicity Synplify
- Exemplar Leonardo

Simulation

- MTI ModelSim/VHDL

Basic Steps

1. Module Generation
 - a) Specify the Design Entry, Vendor, and Behavioral Simulation settings for the project.
2. Prepare for Simulation
 - a) Behavioral simulation
 1. Behavioral module extraction
 2. Declare the library for the behavioral model (MTI ModelSim)
 3. Establish a link to the explicit location of the behavioral models
 4. Analyze the behavioral models
 5. Instantiate the module in the parent design
 - a) Copy the component, instance and configuration declarations into the parent design
 - b) Edit the instantiation template to connect the core to the parent design

6. Create a testbench
 - a) Declare a configuration in the testbench
3. Simulate the Design
 - a) Analyze the parent design and testbench file
 - b) Run the simulation
4. Synthesis and Implementation
 - a) Write out the implementation netlist
 - b) Implement the design in the Xilinx tools

Module Generation

Specify the Design Entry, Vendor, and Behavioral Simulation settings for the project.

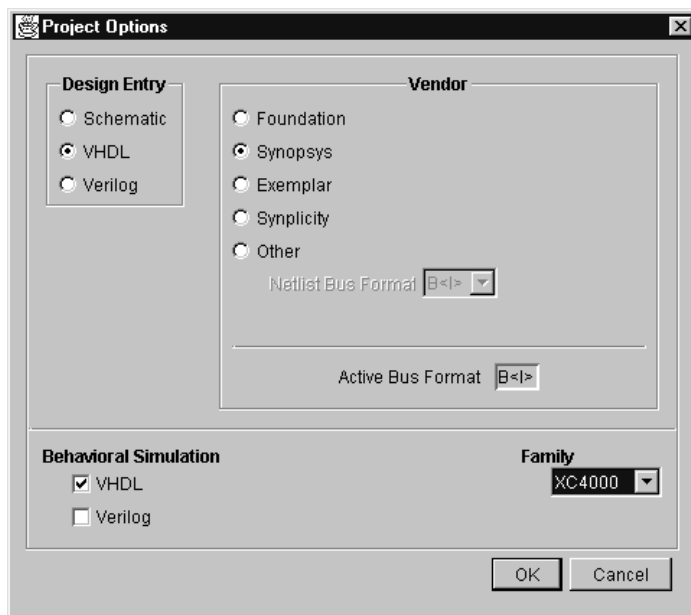


Figure 4-3 VHDL Synopsys Design Flow Settings

From the CORE Generator Project menu, select Project Options and specify the Design Flow as “VHDL”, and specify the synthesis vendor software you will be using to synthesize your VHDL design. This will specify the appropriate EDIF bus delimiter format required for that flow for proper integration with the upper level “parent” implementation netlist written out by the synthesizer for your design.

In the example Figure 4-3 the Design Flow has been set to “VHDL”, and the Vendor has been set to “Synopsys”. The combination of these two settings forces the bus delimiter automatically to the setting of “B<I>”.

Synopsys FPGA Express and FPGA Compiler

- Set Design Flow to “VHDL”, and Vendor to “Synopsys”. This sets the EDIF bus delimiter format to “B<I>”.
- Select the “VHDL” option in the Behavioral Simulation field.

Synplicity Synplify

- Set Design Flow to “VHDL”, and Vendor to “Synplicity”. This sets the EDIF bus delimiter format to “B(I)”.
- Select the “VHDL” option in the Behavioral Simulation field.

Exemplar Leonardo

- Set Design Flow to “VHDL”, and Vendor to “Exemplar”. This sets the EDIF bus delimiter format to “B(I)”.
- Select the “VHDL” option in the Behavioral Simulation field.

Behavioral Simulation

Prepare for Simulation

Note: The following instructions assume that the CORE Generator System v2.1 has been installed to the directory \$XILINX/coregen.

VHDL Model Extraction

Prior to simulating any CORE Generator cores, the source code for their associated behavioral models must be extracted from the CORE Generator installation area to a directory of the user's choosing

(referred to here as “<destination_directory>”). The recommended location of “<destination_directory>” is \$XILINX/vhdl/src, as this is the location of the source code for all other Xilinx VHDL libraries.

The `get_models` utility must be used to extract the models to the <destination_directory>. For a VHDL flow, the `get_models` utility should be run with the `-vhdl` switch to specify that only the vhdl models should be extracted. The full syntax of the command is:

```
get_models -vhdl <destination_directory>
```

Within the specified <destination_directory>, `get_models` creates a separate subdirectory for each vendor (including Xilinx) whose cores exist in the CORE Generator installation. The names of these directories takes the following form:

```
<Vendor>CoreLib
```

Note: The first letter of the vendor name is capitalized

Example

The directory for the Xilinx IP models is “xilinxCoreLib”.

`get_models` extracts all VHDL models provided by a given vendor from the CORE Generator installation area to the appropriate vendor subdirectory.

Note: Installing module updates: Whenever new cores or updates to existing cores are installed into the CORE Generator installation area, the `get_models` utility must be re-invoked to update the repository of VHDL models. (For more information on `get_models`, please refer to the *get_models* section of the *CORE Generator Design Flows* chapter.)

Simulator-specific processing steps

All VHDL simulators require that the VHDL models be analyzed into the simulator's library scheme before simulation can actually proceed.

The analyzed behavioral models provided by a vendor which are extracted by `get_models` must also reside in a library called <Vendor>CoreLib. In the specific case of Xilinx cores, the analyzed behavioral models must reside in a library called `XilinxCoreLib`. The actual physical location (referred to here as <library_directory>) in which the <Vendor>CoreLib resides is not important, but a good choice would be some location which

can be easily accessed by multiple designs and even multiple users for maximum convenience.

Create the XilinxCoreLib library

MTI ModelSim/VHDL

```
cd <library_directory>
vlib xilinxcorelib
```

Note: The name of the analyzed library must be lower case

Establish a link to the compiled behavioral models. To use a vendor's library of compiled behavioral models in a design of your own, a link must be established between your project directory and vendor's library directory.

MTI ModelSim/VHDL

In your project directory, type:

```
vmap xilinxcorelib <library_directory>/xilinxcorelib
```

(Map the logical name of xilinxcorelib to the xilinxcorelib library declared in previous line.)

The above command creates (and also modifies) the MTI modelsim.ini file. This file is read by the ModelSim/VHDL simulator and relates library names to physical locations on a disk or network.

```
vmap xilinxcorelib <full_path_to_"xilinxcorelib">
```

Analyze the behavioral models

Analyze the vendor's VHDL models into this XilinxCoreLib library in the following order (this order is bottom up, starting with the most primitive elements):

```
prims_constants.vhd
prims_comps.vhd
prims_utils.vhd
prims_sim_arch.vhd
ul_utils.vhd
*pack.vhd
c_*.vhd
*.vhd
```

Note: Because this order will change with subsequent IP module updates, please refer to *Xilinx Solution 6250* for the latest information on the recommended compile order for the CORE Generator VHDL behavioral models.

MTI ModelSim/VHDL

To analyze the behavioral models extracted to the XilinxCoreLib subdirectory under <library_directory>, type:

```
vcom -work xilinxcorelib <path_to_extracted_library>/
XilinxCoreLib/prims_constants.vhd
vcom -work xilinxcorelib <path_to_extracted_library>/
XilinxCoreLib/prims_comps.vhd
vcom -work xilinxcorelib <path_to_extracted_library>/
XilinxCoreLib/prims_utils.vhd
vcom -work xilinxcorelib <path_to_extracted_library>/
XilinxCoreLib/prims_sim_arch.vhd
vcom -work xilinxcorelib <path_to_extracted_library>/
XilinxCoreLib/ul_utils.vhd
vcom -work xilinxcorelib <path_to_extracted_library>/
XilinxCoreLib/*pack.vhd
vcom -work xilinxcorelib <path_to_extracted_library>/
XilinxCoreLib/c_*.vhd
vcom -work xilinxcorelib <path_to_extracted_library>/
XilinxCoreLib/*.vhd
```

Note: It is critical that you compile the models in the order specified above; i.e., primitive models before macro level models. Compiling the models in the wrong order will lead to errors in compilation.

The above vlib, vcom and vmap commands need to be performed once for each vendor's library of VHDL models.

Instantiate the module (same procedure for all simulators)

Selecting the “VHDL” checkbox under the Behavioral Simulation option in the CORE Generator Project Options window directs CORE Generator System to generate a VHDL template file (<component_name>.VHO) in addition to an implementation netlist (<component_name>.EDN) whenever a core is generated.

The .VHO template file contains:

- a component declaration
- a component instantiation
- a library declaration
- a configuration “snippet”

These elements need to be copied and pasted into the parent design as follows.

Connect the core to the parent design by editing the instantiation block

Modify the port connections in the instantiation template to reflect the actual connections to the parent design. (See parent design `myadder8_top.vhd` example).

Note: The component declaration and component instantiation block establish a link in the VHDL code to the EDIF implementation netlist for the CORE Generator module. This link is necessary to ensure that the design will be implemented properly after the parent VHDL design has been synthesized. The VHDL instantiation core of the parent design core serves as a placeholder for the core. After the parent design has been synthesized, the core’s EDIF netlist is merged by the Xilinx tools with the rest of the parent design.

Note: The component instantiation contains dummy signal names that must be replaced with the actual signal names in the parent design to which the corresponding pins on the core are to be connected.

The library declaration and the configuration “snippet” in the .VHO VHDL template file are both needed for behavioral simulation only. Notice that both constructs are demarcated in the .VHO file with “`—synopsys translate_off`” and “`—synopsys translate_on`” markers which tell the synthesis tool to ignore the code in between the markers when synthesizing the design. This allows the same code to be used for behavioral simulation and for design synthesis.

This compiler directive is supported by Synopsys FPGA Compiler, Foundation Express, FPGA Express, Exemplar and Synplicity synthesizers.

Note: The specific purpose of the configuration snippet is to establish a link between the parent design and the core's simulation model. A separate, “dummy” .VHD file for the CORE Generator module is not needed. The configuration snippet is used only for behavioral simulation, and has no impact on the synthesis or the implementation processes. To support behavioral simulation of the core, a component declaration for the core, a library declaration for “XilinxCoreLib” or the appropriate <Vendor>CoreLib library, and a VHDL Configuration statement must be added to the parent design and the corresponding configuration snippet for the core from the template file must be embedded in that configuration statement.

It is important to note that a parent design may contain multiple cores, and for each core that is instantiated in the parent design, a corresponding unique configuration “snippet” must be inserted into the parent design's configuration declaration.

Example

The following example illustrates the use of the .VHO template file in a parent design. In this example, an 8-bit registered adder, `myadder8`, is generated by the CORE Generator System and is instantiated in a parent design. The files of interest are the instantiation template file, `myadder8.vho`, and the parent design, `myadder8_top.vhd` and are shown below.

VHDL template file: `myadder8.vho`

```
-----
-- This file was created by the Xilinx CORE Generator --
-- tool, and is (c) Xilinx, Inc. 1998, 1999. No part --
-- of this file may be transmitted to any third party --
-- (other than intended by Xilinx) or used without a --
-- Xilinx programmable or hardware device without --
-- Xilinx's prior written permission. --
-----
-- The following code must appear in the VHDL
-- architecture header:
--- Begin Cut here for COMPONENT Declaration -- COMP_TAG
component myadder8
    port (
        a: IN std_logic_VECTOR(7 downto 0);
```

```
        b: IN std_logic_VECTOR(7 downto 0);
        c: IN std_logic;
        ce: IN std_logic;
        ci: IN std_logic;
        clr: IN std_logic;
        s: OUT std_logic_VECTOR(8 downto 0));
end component;
-- COMP_TAG_END ----- End COMPONENT Declaration -----

-- The following code must appear in the VHDL
-- architecture body. Substitute your own instance name
-- and net names.
--- Begin Cut here for INSTANTIATION Template -- INST_TAG
your_instance_name : myadder8
    port map (
        a => a,
        b => b,
        c => c,
        ce => ce,
        ci => ci,
        clr => clr,
        s => s);
-- INST_TAG_END ----- End INSTANTIATION Template -----
-- The following code must appear above the VHDL
-- configuration declaration. An example is given at
-- the end of this file.

--- Begin Cut here for LIBRARY Declaration --- LIB_TAG
-- synopsys translate_off

Library XilinxCoreLib;
-- synopsys translate_on

-- LIB_TAG_END ----- End LIBRARY Declaration -----
-- The following code must appear within the VHDL
-- top-level configuration declaration. Ensure that
-- the translate_off/on compiler directives are correct
-- for your synthesis tool(s).
```

```
--- Begin Cut here for CONFIGURATION snippet --- CONF_TAG

-- synopsys translate_off

    for all : myadder8 use entity
XilinCoreLib.adreVHT(behavioral)
        generic map(
            Signed => 1,
            Input_Width => 8);
        end for;
-- synopsys translate_on
-- CONF_TAG_END ----- End CONFIGURATION snippet -----

-----
-- Example of configuration declaration...
-----

--
-- <Insert LIBRARY Declaration here>
--
-- configuration <cfg_my_design> of <my_design> is
--     for <my_arch_name>
--         <Insert CONFIGURATION Declaration here>
--     end for;
-- end <cfg_my_design>;
--
-- If this is not the top-level design then in the next
-- level up, the following text should appear
-- at the end of that file:
--
-- configuration <cfg> of <next_level> is
--     for <arch_name>
--         for all : <my_design> use configuration
-- <cfg_my_design>;
--         end for;
--     end for;
-- end <cfg>;
--
```

Next, we take a look at the parent design, `myadder8_top.vhd`. Notice that the component declaration, the instantiation (with dummy signal names replaced with actual signal names) and the configuration snippet have been cut-and-pasted from `myadder8.vho`.

VHDL parent design file: `myadder8_top.vhd`

```
library IEEE;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_1164.all;
ENTITY myadder8_top IS
    PORT (ap : IN std_logic_vector(7 downto 0);
          bp : IN std_logic_vector(7 downto 0);
          ck: IN std_logic ;
          cep: IN std_logic;
          cip: IN std_logic;
          clrp: IN std_logic;
          sp: OUT std_logic_VECTOR (8 downto 0));
END myadder8_top;
ARCHITECTURE use_core of myadder8_top IS
```

```
-----
---- The MYADDER8 core is used in this design. The core
---- must be declared via a 'component declaration';
---- myadder8.vho provides the component declaration
---- which is cut-and-pasted into the design as
---- shown below.
-----
```

```
component myadder8
    port (
        a: IN std_logic_VECTOR(7 downto 0);
        b: IN std_logic_VECTOR(7 downto 0);
        c: IN std_logic;
        ce: IN std_logic;
        ci: IN std_logic;
        clr: IN std_logic;
        s: OUT std_logic_VECTOR(8 downto 0));
end component;
```

```
BEGIN
-----
---- The core is instantiated into this design.
---- myadder8.vho provides an instantiation
---- template which must be modified
---- so that it reflects actual signals used in the
---- design, establishing the connectivity between the
---- core and other logic at this level. The instance
---- of the core must also be given an actual label to
---- replace the dummy "your_instance_name" tag. In this
---- example, it is replaced by "myadder8".
-----
myadder8_1      : myadder8
                  port map (
                      a => ap,
                      b => bp,
                      c => ck,
                      ce => cep,
                      ci => cip,
                      clr => clrp,
                      s => sp );
end use_core;

-----
A partial configuration statement is found in
---- myadder8.vho. It contains information necessary to
---- link the behavior of the core with the its
---- instantiation. This configuration section from
---- myadder8.vho must be reproduced in this parent
---- design, embedded within the configuration statement
---- of parent design. NOTE: For each core that is
---- instantiated in this design, a unique partial
---- configuration statement must be included.
-----
-- synopsys translate_off
CONFIGURATION cfg_myadder8_top OF myadder8_top IS
    FOR use_core
```

```
for all : myadder8 use entity
XilinxCoreLib.myadder8firVHT(behavioral)

    generic map(
        signed => 1,
        input_width => 8);
    end for;
end for;
end cfg_myadder8_top;
-- synopsys translate_on
```

Create the Testbench

To simulate a parent design containing the MYADDER8 core, a testbench, `testbench.vhd`, is written that includes an instantiation of the parent design. The testbench also includes a configuration statement that reminds the simulator to reference the configuration statement in the parent design. The testbench should also contain stimulus to activate the adder. Part of the testbench file used to simulate this design is shown below; with the exception that the section that would have contained simulation stimulus is omitted.

VHDL Testbench file: `testbench.vhd`

```
library IEEE;
use IEEE.std_logic_1164.ALL;
ENTITY testbench is
END testbench;

ARCHITECTURE simulate OF testbench IS
-----
---- The parent design, myadder8_top, is instantiated
---- in this testbench. Note the component
---- declaration and the instantiation.
-----
COMPONENT myadder8_top
    PORT (
        ap : IN std_logic_vector(7 downto 0);
        bp : IN std_logic_vector(7 downto 0);
        ck: IN std_logic ;
        cep: IN std_logic;
        cip: IN std_logic;
```

```
        clrp: IN std_logic;
        sp: OUT std_logic_VECTOR (8 downto 0));
END myadder8_top;

SIGNAL a_data_input : std_logic_vector(7 DOWNTO 0);
SIGNAL b_data_input  : std_logic_vector(7 DOWNTO 0);
SIGNAL clock         : std_logic;
SIGNAL clock_enable  : std_logic;
SIGNAL carry_in      : std_logic;
SIGNAL clear         : std_logic;
SIGNAL sum           : std_logic (8 DOWNTO 0);
```

```
BEGIN
```

```
  uut: myadder8_top
```

```
    PORT MAP (
```

```
  ap => a_data_input,
    bp => b_data_input,
    ck => clock,
    cep => clock_enable,
    cip => carry_in,
    clrp=> clear,
    sp => sum);
```

```
  stimulus: PROCESS
```

```
    BEGIN
```

```
-----
-- Provide stimulus in this section. (not shown here)
-----
```

```
        wait;
        end process; -- stimulus
```

```
END simulate;
```

```
-----
The configuration, cfg_testbench, of the testbench,
---- reminds the simulator to refer to the configuration
---- statement in the parent design. Note that "work" was
---- was the default library into which the testbench
---- and the parent design were analyzed.
```

```
-----  
CONFIGURATION cfg_testbench OF testbench IS  
FOR simulate  
  for all : myadder8_top  
    use configuration work.cfg_myadder8_top;  
  end for;  
END for;  
END cfg_testbench;
```

Design Simulation

Analyze the parent design and testbench file

Before Model Technology's simulation tools can be used to simulate the design, the parent design and the testbench must be analyzed. Unlike the earlier step in which all the behavioral models were analyzed into XilinxCoreLib (or <Vendor>CoreLib), these design files may be analyzed (using the `vcom` command) into a local, default, work library (created using the `vlib` command).

MTI ModelSim

From the <project_directory>, type:

```
vlib work  
vcom myadder8_top.vhd  
vcom testbench.vhd
```

Invoke the simulator

The simulator may now be invoked:

```
vsim cfg_testbench
```

where, `cfg_testbench` corresponds to the name of the VHDL configuration declared in the testbench. This will load the testbench, the parent design and the simulation model of the `myadder8` core, stored in the location referenced by `XilinxCoreLib`.

Synthesis and Implementation

Synthesize the design using black-box methodology

The parent design containing one or more cores that must be synthesized. During the synthesis process, the synthesizer must be directed

to treat any instantiated cores as “black-boxes”, because the logic for each core is specified only in its EDIF implementation netlist <component_name.EDN>, not in a VHDL file.

Synopsys FPGA Compiler

Apply the “don’t_touch” attribute to the module via the Synopsys compile script.

Synopsys FPGA Express

Make sure you do NOT read in a separate .VHD or EDIF file for the CORE Generator module. FPGA Express will automatically treat the module as a “black box”.

Synplicity Synplify v5.1.2 and later

Attach a “black_box” attribute to component declaration for the CORE Generator module as follows (this attribute is optional, but will prevent Synplicity from issuing warnings about “black box” modules):

```
-- VHDL black box attribute example
attribute black_box : boolean;
component myadder8
  port (
    a: IN std_logic_VECTOR(7 downto 0);
    b: IN std_logic_VECTOR(7 downto 0);
    c: IN std_logic;
    ce: IN std_logic;
    ci: IN std_logic;
    clr: IN std_logic;
    s: OUT std_logic_VECTOR(8 downto 0));
end component;
attribute black_box of myadder8 : component is true;
```

In addition, make sure you do NOT read in a separate .VHD or EDIF file for the CORE Generator module. Synplify will automatically treat the module as a “black box”.

Exemplar Leonardo (v1998.2)

Simply make sure that you do NOT read in a separate .VHD or EDIF file for the CORE Generator module. Leonardo will automatically treat the module as a black box.

Write out the implementation netlist for the synthesized design

After the parent design has been synthesized, you must write out its implementation netlist using the synthesis tool. Depending on the synthesis tool being used and the target architecture, this implementation netlist may be an EDIF or XNF file, or a set of EDIF or XNF files.

Note: CORE Generator System breaks buses out into their component bits when writing out the EDIF implementation netlist for a module. This formerly created pin mismatch problems with the upper level EDIF written out by some synthesis tools. However, starting with the 1.5 release of the Xilinx Implementation tools, EDIF2NGD will automatically resolve connections between bus nets written out in bus format (for example, address<7:0>) in a parent EDIF netlist, and bus nets written out as individual bits in a lower level EDIF.

Synopsys FPGA Compiler

No special instructions. FPGA Compiler will write out an XNF file for 4K designs, and EDIF for Virtex designs.

Synopsys FPGA Express

No special instructions. FPGA Express will write out an XNF file for 4K designs, and EDIF for Virtex designs.

Synplicity Synplify v5.1.2and later

Synplify can write out either XNF or EDIF netlists for both XC4000 and Virtex designs. EDIF format is preferred. No other special instructions.

Exemplar Leonardo (v1998.2)

Leonardo can write out either XNF or EDIF netlists for both XC4000 and Virtex designs. EDIF format is preferred. No other special instructions.

Implement the Design

The implementation netlists for each of the cores in the parent design are merged in with the main design when the ngdbuild program (the “Translate” stage of the Xilinx Design Manager) is run on the top level parent design during design implementation. For the merging of the netlists to succeed, you must make sure that the CORE Generator .EDN EDIF netlist(s) for the generated module or modules are located in the same directory as the top level EDIF netlist for the synthesized design, or you must run NGDBUILD with the `-sd` option, specifying the location of the directory containing the CORE Generator EDN files.

