

CORE Generator Guide

Introduction

Getting Started

Using the CORE Generator

***Understanding CORE
Generator Design Flows***

***Understanding the HDL
Design Flow***

***Troubleshooting the CORE
Generator System***



The Xilinx logo shown above is a registered trademark of Xilinx, Inc.

ASYL, FPGA Architect, FPGA Foundry, NeoCAD, NeoCAD EPIC, NeoCAD PRISM, NeoROUTE, Timing Wizard, TRACE, XACT, XILINX, XC2064, XC3090, XC4005, XC5210, and XC-DS501 are registered trademarks of Xilinx, Inc.



The shadow X shown above is a trademark of Xilinx, Inc.

All XC-prefix product designations, A.K.A Speed, Alliance Series, AllianceCORE, BITA, CLC, Configurable Logic Cell, CoolRunner, CORE Generator, CoreLINX, Dual Block, EZTag, FastCLK, FastCONNECT, FastFLASH, FastMap, Fast Zero Power, Foundation, HardWire, IRL, LCA, LogiBLOX, Logic Cell, LogiCORE, LogicProfessor, MicroVia, MultiLINX, PLUSASM, PowerGuide, PowerMaze, QPro, RealPCI, RealPCI 64/66, SelectI/O, SelectRAM, SelectRAM+, Silicon Xpresso, Smartguide, Smart-IP, SmartSearch, Smartspec, SMARTSwitch, Spartan, TrueMap, UIM, VectorMaze, VersaBlock, VersaRing, Virtex, WebFitter, WebLINX, WebPACK, XABEL, XACTstep, XACTstep Advanced, XACTstep Foundry, XACT-Floorplanner, XACT-Performance, XAM, XAPP, X-BLOX, X-BLOX plus, XChecker, XDM, XDS, XEPLD, Xilinx Foundation Series, XPP, XSI, 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, Inc. 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, Inc. 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, Inc. will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx, Inc. 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; 4,855,619; 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; 5,455,525; 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; 5,504,439; 5,506,518; 5,506,523; 5,506,878; 5,513,124; 5,517,135; 5,521,835; 5,521,837; 5,523,963; 5,523,971; 5,524,097; 5,526,322; 5,528,169; 5,528,176; 5,530,378; 5,530,384; 5,546,018; 5,550,839; 5,550,843; 5,552,722; 5,553,001; 5,559,751; 5,561,367; 5,561,629; 5,561,631; 5,563,527; 5,563,528; 5,563,529; 5,563,827; 5,565,792; 5,566,123; 5,570,051; 5,574,634; 5,574,655; 5,578,946; 5,581,198; 5,581,199; 5,581,738; 5,583,450; 5,583,452; 5,592,105; 5,594,367; 5,598,424; 5,600,263; 5,600,264; 5,600,271; 5,600,597; 5,608,342; 5,610,536; 5,610,790; 5,610,829; 5,612,633; 5,617,021; 5,617,041; 5,617,327; 5,617,573; 5,623,387; 5,627,480; 5,629,637; 5,629,886; 5,631,577; 5,631,583; 5,635,851; 5,636,368; 5,640,106; 5,642,058; 5,646,545; 5,646,547; 5,646,564; 5,646,903; 5,648,732; 5,648,913; 5,650,672; 5,650,946; 5,652,904; 5,654,631; 5,656,950; 5,657,290; 5,659,484; 5,661,660; 5,661,685; 5,670,896; 5,670,897; 5,672,966; 5,673,198; 5,675,262; 5,675,270; 5,675,589; 5,677,638; 5,682,107; 5,689,133; 5,689,516; 5,691,907; 5,691,912; 5,694,047; 5,694,056; 5,724,276; 5,694,399; 5,696,454; 5,701,091; 5,701,441; 5,703,759; 5,705,932; 5,705,938; 5,708,597; 5,712,579; 5,715,197; 5,717,340; 5,719,506; 5,719,507; 5,724,276; 5,726,484; 5,726,584; 5,734,866; 5,734,868; 5,737,234; 5,737,235;

5,737,631; 5,742,178; 5,742,531; 5,744,974; 5,744,979; 5,744,995; 5,748,942; 5,748,979; 5,752,006; 5,752,035; 5,754,459; 5,758,192; 5,760,603; 5,760,604; 5,760,607; 5,761,483; 5,764,076; 5,764,534; 5,764,564; 5,768,179; 5,770,951; 5,773,993; 5,778,439; 5,781,756; 5,784,313; 5,784,577; 5,786,240; 5,787,007; 5,789,938; 5,790,479; 5,790,882; 5,795,068; 5,796,269; 5,798,656; 5,801,546; 5,801,547; 5,801,548; 5,811,985; 5,815,004; 5,815,016; 5,815,404; 5,815,405; 5,818,255; 5,818,730; 5,821,772; 5,821,774; 5,825,202; 5,825,662; 5,825,787; 5,828,230; 5,828,231; 5,828,236; 5,828,608; 5,831,448; 5,831,460; 5,831,845; 5,831,907; 5,835,402; 5,838,167; 5,838,901; 5,838,954; 5,841,296; 5,841,867; 5,844,422; 5,844,424; 5,844,829; 5,844,844; 5,847,577; 5,847,579; 5,847,580; 5,847,993; 5,852,323; 5,861,761; 5,862,082; 5,867,396; 5,870,309; 5,870,327; 5,870,586; 5,874,834; 5,875,111; 5,877,632; 5,877,979; 5,880,492; 5,880,598; 5,880,620; 5,883,525; 5,886,538; 5,889,411; 5,889,413; 5,889,701; 5,892,681; 5,892,961; 5,894,420; 5,896,047; 5,896,329; 5,898,319; 5,898,320; 5,898,602; 5,898,618; 5,898,893; 5,907,245; 5,907,248; 5,909,125; 5,909,453; 5,910,732; 5,912,937; 5,914,514; 5,914,616; 5,920,201; 5,920,202; 5,920,223; 5,923,185; 5,923,602; 5,923,614; 5,928,338; 5,931,962; 5,933,023; 5,933,025; 5,933,369; 5,936,415; 5,936,424; 5,939,930; 5,942,913; 5,944,813; 5,945,837; 5,946,478; 5,949,690; 5,949,712; 5,949,983; 5,949,987; 5,952,839; 5,952,846; 5,955,888; 5,956,748; 5,958,026; 5,959,821; 5,959,881; 5,959,885; 5,961,576; 5,962,881; 5,963,048; 5,963,050; 5,969,539; 5,969,543; 5,970,142; 5,970,372; 5,971,595; 5,973,506; 5,978,260; 5,986,958; 5,990,704; 5,991,523; 5,991,788; 5,991,880; 5,991,908; 5,995,419; 5,995,744; 5,995,988; 5,999,014; 5,999,025; 6,002,282; and 6,002,991; 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, Inc. 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, Inc. 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 1991-2000 Xilinx, Inc. All Rights Reserved.

About This Manual

This manual describes the Xilinx CORE Generator™ Tool, which is used for parameterizing cores optimized for Xilinx FPGAs.

Note This Xilinx software release is certified Year 2000 compliant.

Manual Contents

This manual covers the following topics:

- Chapter 1, “Introduction”—Introduces the Xilinx CORE Generator System by describing the process for installation and how to obtain new and updated COREs.
- Chapter 2, “Getting Started”—Provides information for setting up your environment and installing the CORE Generator.
- Chapter 3, “Using the CORE Generator”—Describes the CORE browser, customizing a CORE, updating COREs, and integrating the CORE Generator into applications.
- Chapter 4, “Understanding CORE Generator Design Flows”—Describes how to integrate a CORE Generator module into a design through the use of various design flows; schematic and HDL.
- Chapter 5, “Understanding the HDL Design Flow”—Describes the elements and procedures in a HDL design flow.
- Appendix A, “Troubleshooting the Core Generator System”—Contains solutions and resources for using the CORE Generator System.

Additional Resources

For additional information, go to <http://support.xilinx.com>. The following table lists some of the resources you can access from this Web site. You can also directly access these resources using the provided URLs.

Resource	Description/URL
Tutorials	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://support.xilinx.com/support/techsup/tutorials/index.htm
Answers Database	Current listing of solution records for the Xilinx software tools Search this database using the search function at http://support.xilinx.com/support/searchtd.htm
Application Notes	Descriptions of device-specific design techniques and approaches http://support.xilinx.com/apps/appsweb.htm
Data Book	Pages from <i>The Programmable Logic Data Book</i> , which contain device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging http://support.xilinx.com/partinfo/databook.htm
Xcell Journals	Quarterly journals for Xilinx programmable logic users http://support.xilinx.com/xcell/xcell.htm
Technical Tips	Latest news, design tips, and patch information for the Xilinx design environment http://support.xilinx.com/support/techsup/journals/index.htm

Conventions

This manual uses the following conventions. An example illustrates each 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 required.

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 loc2locn;*

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.

Contents

About This Manual

Manual Contents	v
Additional Resources	vi

Conventions

Typographical.....	vii
Online Document	viii

Chapter 1 Introduction

CORE Generator System.....	1-1
CORE Generator Components	1-4
New and Updated Cores.....	1-4
System Requirements and Installation Information.....	1-4
Additional Resources	1-4

Chapter 2 Getting Started

Starting the CORE Generator System	2-1
CORE Generator System Installation Requirements.....	2-2
Setting Preferences	2-2
Setting Up Projects	2-3
Creating a New Project	2-4
Opening an Existing Project.....	2-4
Selecting Design Entry Options	2-6
Schematic Design Environment.....	2-6
HDL Synthesis Design Environment.....	2-6
Selecting Target XILINX FPGA Family Options.....	2-7
Changing Project Design Entry Options.....	2-7

Installing Setup Files	2-8
coregen.prj	2-8
corelib.xml	2-8
.coregen.prj	2-9
Using the Web Browser and the PDF Viewer	2-9

Chapter 3 Using the CORE Generator

Using the CORE Browser	3-1
Customizing a Core.....	3-4
Displaying the CoreViewer.....	3-5
Naming CORE Generator Modules	3-7
Using the Generate, Cancel, and Data Sheet Buttons	3-8
Illegal or Invalid Values	3-8
.COE Files.....	3-8
Specifying Command Files	3-11
coregen.ini/coregen_user_name.ini.....	3-12
User-Generated Command Files	3-12
XCO Files.....	3-13
coregen.log	3-14
Generating Cores in Batch Mode.....	3-14
Defining CORE Generator Command Line Options.....	3-15
-b command_file_name.....	3-15
-i coregen_ini_file_name	3-15
-p project_path	3-15
-q polling_dir_path.....	3-15
-h.....	3-16
Listing the CORE Generator Commands.....	3-17
Listing the CORE Generator Global Properties	3-18
Listing Project Properties	3-19
Updating Cores in the CORE Generator.....	3-20
Downloading New Cores	3-21
Updating a Core Version in an Existing Project	3-21
Understanding the Update Project Cores Menu	3-22
Removing Cores	3-23
Using the get_models Command	3-24
Integrating CORE Generator into Applications	3-27
Polling Mode	3-27
Output Polling Files.....	3-28
Input Polling Files.....	3-28
ASY and XSF Files	3-29
Listing Inputs and Outputs Files.....	3-29

Chapter 4 Understanding CORE Generator Design Flows

Understanding CORE Generator Design Flow Basics	4-1
Describing the CORE Generator Schematic Design Flow	4-2
Starting a Schematic Design Flow with Viewlogic	4-3
Creating a Viewlogic Project	4-4
Creating Output Files	4-6
Foundation Design Flow	4-8
Foundation ISE Design Flow	4-8
Mentor Design Flow	4-8
Cadence Design Flow	4-8
Describing the HDL Behavioral Model Delivery System Features ..	4-8
XilinxCoreLib Simulation Library	4-9
coredb	4-9
Instantiation Template Files	4-9
Support for Unused Optional Pins	4-10
verilog_analyze_order File	4-10
vhdl_analyze_order File	4-10
Using the CORE Generator Verilog Design Flow Procedure	4-11
Using Instantiation Templates	4-12
Using a .VEO Instantiation Template File	4-12
Verilog Instantiation Template for an 8-Bit Adder	4-12
Using the CORE Generator VHDL Design Flow Procedure	4-14
Using a .VHO Instantiation Template File	4-15
VHDL Instantiation Template for an 8-Bit Adder	4-15

Chapter 5 Understanding the HDL Design Flow

Using the HDL Behavioral Model Delivery System	5-1
Understanding the Verilog HDL Design Flow	5-2
Describing the Verilog Design Flow Procedure	5-2
Implementation Using Cadence Verilog-XL and MTI Model Sim/VLOG	5-17
Understanding the VHDL HDL Design Flow	5-18
Describing the VHDL Design Flow Procedure	5-19

Appendix A Troubleshooting the Core Generator System

Finding Solutions	A-1
Additional Resources	A-2
AllianceCORE Modules	A-2
Obtaining Customer Support	A-2

Introduction

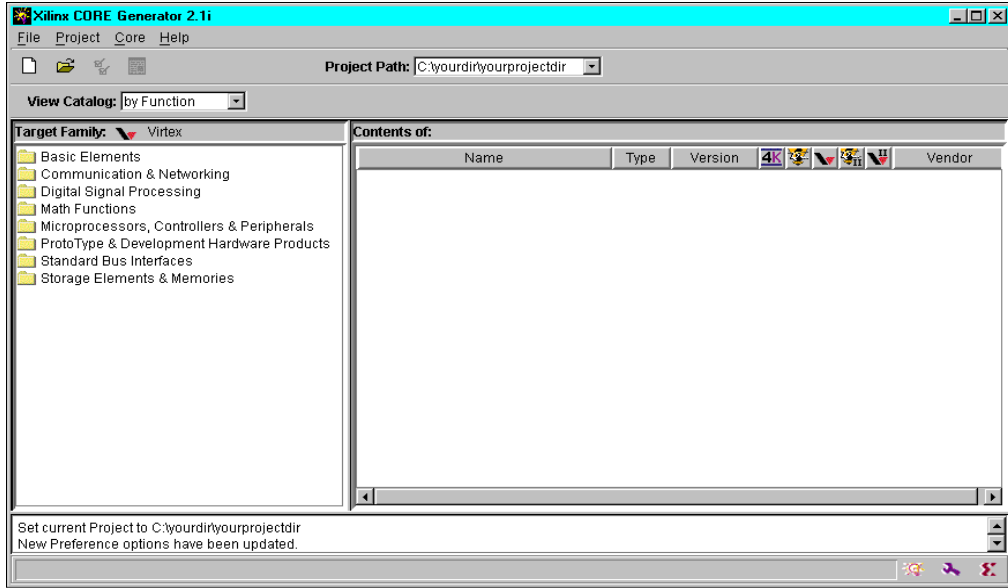
This chapter introduces the Xilinx CORE Generator System, an easy to use design tool that delivers parameterizable COREs optimized for Xilinx FPGAs.

The following topics are included in this chapter:

- “CORE Generator System”
- “CORE Generator Components”
- “New and Updated Cores”
- “System Requirements and Installation Information”
- “Additional Resources”

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, as shown in the following figure:

**Figure 1-1 CORE Generator GUI**

The Xilinx CORE Generator System provides you 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.

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 functional type into folders that expand or contract on demand. Detailed information on each core is included in a specification or data sheet (Acrobat format), which is easily accessed by clicking on the Datasheet button in the core customization 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 the following items:

- Functional information
- Area and performance data for some cores

- Pinouts and interface signal names
- Details on how to use the core in an application, making it easy for you to determine whether a core meets your design requirements

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 deliver high levels of performance and area efficiency. This is accomplished by using Xilinx's core-friendly FPGA architectures and Xilinx Smart-IP™ technology. Smart-IP technology leverages the following items:

- 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 the following:

- 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
- 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 the following:

- A customized EDIF netlist
- Verilog® or VHDL behavioral simulation models
- Verilog or VHDL Instantiation templates
- Foundation™ or Viewlogic® schematic symbols

CORE Generator Components

The Xilinx CORE Generator system consists of the following three distinct products:

- CORE Generator application
- Acrobat™ Reader application
- JAVA™ Runtime Support

New and Updated Cores

New cores can be downloaded and easily added to the CORE Generator from the Xilinx Website at either

<http://www.xilinx.com/ipcenter/coregen/updates.htm>

or at

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

Please review the CORE Generator Web page before starting a new design. You need to verify that you have the latest version of each core and core datasheet.

System Requirements and Installation Information

See the 3.1i Release Notes for information on system requirements and installation instructions for the CORE Generator System or the *Xilinx Alliance Quick Start Guide* or *Foundation Quick Start Guide*.

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

For Viewlogic users, 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

The following section details additional online documentation resources and how to access the information.

Links to the IP Center are available from the CORE Generator Help Menu with the following path:

Help → Help on the Web → IP Center

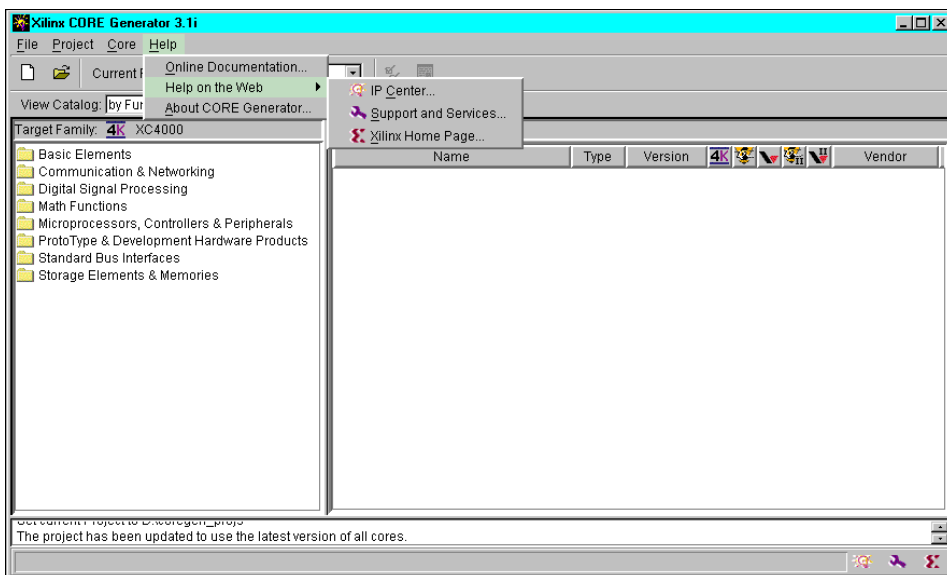


Figure 1-2 Help Menu

For an overview of the supported design flows, refer to Chapter 4, “Understanding CORE Generator Design Flow Basics” in this manual.

Getting Started

This chapter describes the various elements of the CORE Generator System. Review this information before starting a design using the cores offered with the CORE Generator System. The following sections are described in this chapter:

- “Starting the CORE Generator System”
- “Setting Up Projects”
- “Opening an Existing Project”
- “Selecting Design Entry Options”
- “Selecting Target XILINX FPGA Family Options”
- “Changing Project Design Entry Options”
- “Installing Setup Files”
- “Using the Web Browser and the PDF Viewer”

Starting the CORE Generator System

This section describes the functions performed by the user in initiating, designing, and maintaining core designs in Xilinx CORE Generator System’s GUI environment.

CORE Generator System Installation Requirements

To install the CORE Generator in either your Windows or UNIX workstation environments, do one of the following:

- (Windows) Select **Start → Programs → Xilinx Alliance Series 2.1i → Accessories → CORE Generator System**.
- (UNIX Workstation) At a UNIX shell prompt, type **coregen**. This starts the CORE Generator System. Make sure your environment is setup to run the Xilinx software as specified in the *Xilinx Alliance Quick Start Guide* or *Foundation Quick Start Guide*. The two required settings are 1) *XILINX* variable, set to your Xilinx installation directory, and 2) *PATH* variable, set to *\$XILINX/bin/platform*. The platform is either *sol* or *hp*.

Note For detailed information on CORE Generator installation for both the PC and the Unix Workstation, refer to the *Xilinx Alliance Quick Start Guide* or *Foundation Quick Start Guide*.

The CORE Generator application is installed from one of the following main Xilinx software release CDs:

- Alliance 3.1i
- Foundation 3.1i

Setting Preferences

Your preferences are set through the preferences dialog box and are maintained on a per user basis, as shown in the following figure:

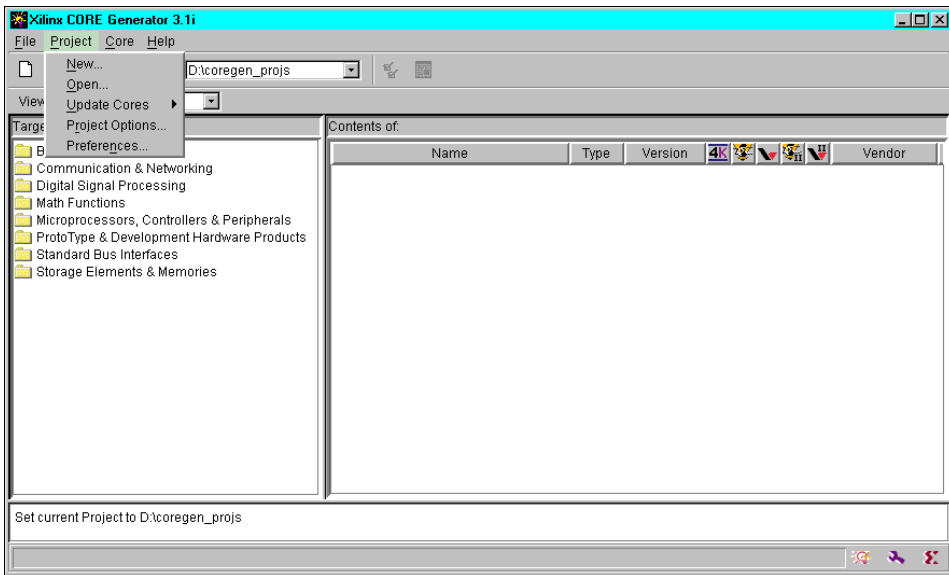


Figure 2-1 Preference Dialog Box

The location for Preferences on various platforms are as follows:

- (Windows) Preferences are stored in the registry and written to the Windows registry.
- (UNIX workstation) Preferences are stored in your home directory in the file, `coregen.prf`.

Setting Up Projects

The following section describes creating new projects, opening up existing projects, selecting design entry options, and other similar topics.

Creating a New Project

This next section describes the procedure for creating a new project using the Xilinx CORE Generator System.

1. Select **Project** → **New**

In the New Project screen, you can type the path to the new project directory into the **Directory** text field or you can click on the **Browse** button and navigate to it.

Note The Xilinx CORE Generator System is designed to operate within the directory structure of a pre-existing design entry environment. Because of this, the CORE Generator System does not create directories. You must make sure a directory exists before browsing to it.

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

2. Specify your Electronic Design Automation (EDA) Vendor from the following selection: **Foundation**, **Viewlogic**, **Cadence**, **Mentor**, or **Other**).

If you select **Other**, you need to specify the Netlist Bus Format for individual bus bits. The B represents the name of the bus and the I represents the bus index, for example B<I>, B[I], or BI. Selecting any of the other vendors automatically sets the Netlist Bus Format setting to the correct value for that vendor.

Opening an Existing Project

You can track the actual number of previously opened projects through the **Project** → **Preferences** option in the main CORE Generator GUI.

1. Select a project from the Project Path list in the dialog box. Select the CORE Generator project that you want to work in and click **OK**.
2. You may also place a check mark in the **Always Open Last Project** check box in the following figure. Selecting this box causes the CORE Generator System to bypass the CORE Generator dialog box, and to return to the last project worked on.

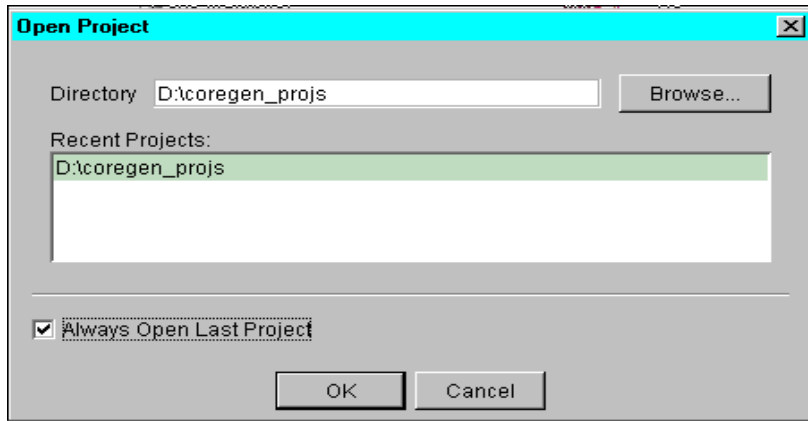


Figure 2-2 Open Project Box

Note When starting up the CORE Generator System, select **Open Project** and deselect the **Always Open Last Project** check box in the **Open Project** dialog box. This restores the launching of the **CORE Generator** project dialog box

For each user, the CORE Generator remembers the last *n* projects that you opened. You can select a project from this list or browse to any valid CORE Generator project. Each project maintains a list of the cores visible to that project and their version. If a new IP has been added to the repository or removed from it, the CORE Generator pops up a dialog asking if you want to update the list of visible cores for that project.

When a project is opened by CORE Generator, it is locked to prevent other users from working in the same project and potentially overwriting files. If another user tries to open the project, they receive a dialog showing who has the project locked. The lock can be removed from this file. If the original CORE Generator session loses its lock, then the next time you attempt to generate a core, you will receive a dialog showing who overrode the lock.

Selecting Design Entry Options

From the Project Options dialog, select the Design Entry that you would like to use. Selecting one of the following entries shows the corresponding vendor(s) supported in the Vendor box:

Schematic Design Environment

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

- Viewlogic
- Foundation
- Mentor®
- Cadence®

Note Limited Cadence support is currently available as described in the “Cadence Design Flow” section.

HDL Synthesis Design Environment

The CORE Generator design environment currently supports the following Hardware Descriptive Language (HDL) Synthesis tools, which consist of VHDL or Verilog:

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

When you chose VHDL or Verilog, the corresponding instantiation template is generated in `module_name.vho` or `module_name.veo`. These files contain commented HDL code that can be used to instantiate a CORE Generator module in an HDL design, and also contain code that supports behavioral simulation. If you select VHDL as the design entry, then a `.VHO` file is generated. If you select Verilog as the design entry, then a `.VEO` file is generated. Vendor specific files may also be generated for schematic symbols and other uses.

Based on the Vendor chosen, the output EDIF netlist contains the appropriate bus delimiter for the module (), <>, [], or none. This is necessary so that the ports in the CORE Generator module match the port references in the EDIF netlist for your top level design. The vendor option also controls the generation of vendor specific pin and symbol files.

Selecting Target XILINX FPGA Family Options

The Xilinx CORE Generator System tailors the cores to the selected Target Family setting. All cores are optimized to the selected Xilinx architecture and do not work if integrated into a design targeted to a different Xilinx FPGA family. Cores, that were targeted to the Spartan architecture when they were generated, do not work if placed in a Virtex design. You need to select the Target Family based on the Xilinx architecture that you are targeting. Changing architectures requires you to regenerate any cores you have already created.

After you have selected all the project options, click **OK**. The Xilinx CORE Generator System initializes the new project. This initialization may take several seconds. A coregen.prj file is written to the new project. The coregen.prj file contains a record of all installed cores at the time of project creation and their available versions. In order for the list of cores available to be displayed, you need to specify a valid CORE Generator project.

Changing Project Design Entry Options

You may change the Project Design Entry Options as follows:

1. Select **Project** → **Project Options** menu.

This opens a Project Options dialog box. You can change any of the Design Entry, Vendor, Behavioral Simulation, and Family options.

2. Click **OK** when you have finished modifying these options.

Note Changing the project options only affects new cores that you generate. Cores created before making the project changes still reflect the old options. Regenerate any cores that need to inherit the new project options.

Installing Setup Files

The following section describes in some detail the setup files in the CORE Generator. The setup file is required to properly configure your CORE Generator session.

coregen.prj

The `coregen.prj` is the CORE Generator project file and is written to your home directory on your UNIX workstation. 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.

The `coregen.prj` file 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.

corelib.xml

The CORE catalog file is called `corelib.xml` and is located in `$XILINX/coregen/ip`. You build this catalog by scanning each of the cores found in `$XILINX/coregen/ip`. This catalog is used by CORE Generator at runtime to identify and locate all the cores that are present in a CORE Generator software installation. You can also update the catalog manually using the `coredb` utility.

The `corelib.xml` is present in the initial installation and updated with the installation of each ip update. There is a utility `coredb` that you can use to regenerate it if necessary. CORE Generator generates `corelib.xml` if it is not present or out of synch with the ip repository. If the `corelib.xml` needs to be built and you cannot write to `$XILINX/coregen/ip`, CORE Generator can run but startup time is impacted by trying to rebuild `corelib.xml` each time.

.coregen.prf

The .coregen.prf is the Xilinx CORE Generator preferences file for UNIX workstations. This is an ASCII option settings file that records various user specific settings for the CORE Generator GUI. This file 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`. For example,

```
AlwaysOpenLastProject=true
```

Each Property Name represents a particular property within the Xilinx CORE Generator, and the corresponding Value Field is the value to be applied to that property. Your preference settings are stored in your home directory on a PC vendors platform. This is recorded in the Windows registry. The name of your preference file should be

During start-up, and after any optional coregen.ini file is read (Workstations only), CORE Generator System searches the preferences directory for a .coregen.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.

`.coregen.prf`.

The first time you start up the CORE Generator System, you will not see a preference file. The preference file, .coregen.prf, is created the first time you exit out of the CORE Generator System. The file is automatically written to `$XILINX/coregen/preferences` when you exit the CORE Generator application, based on settings you have specified during a project session.

Using the Web Browser and the PDF Viewer

Another feature of the Xilinx CORE Generator System, is the ability to link to sites on the Web. You can click to the Xilinx CORE Generator System Home page or the Xilinx support.xilinx.com site. You can also link to the AllianceCORE partner Websites.

The Xilinx CORE Generator System also provides all core datasheets in Adobe Acrobat PDF format.

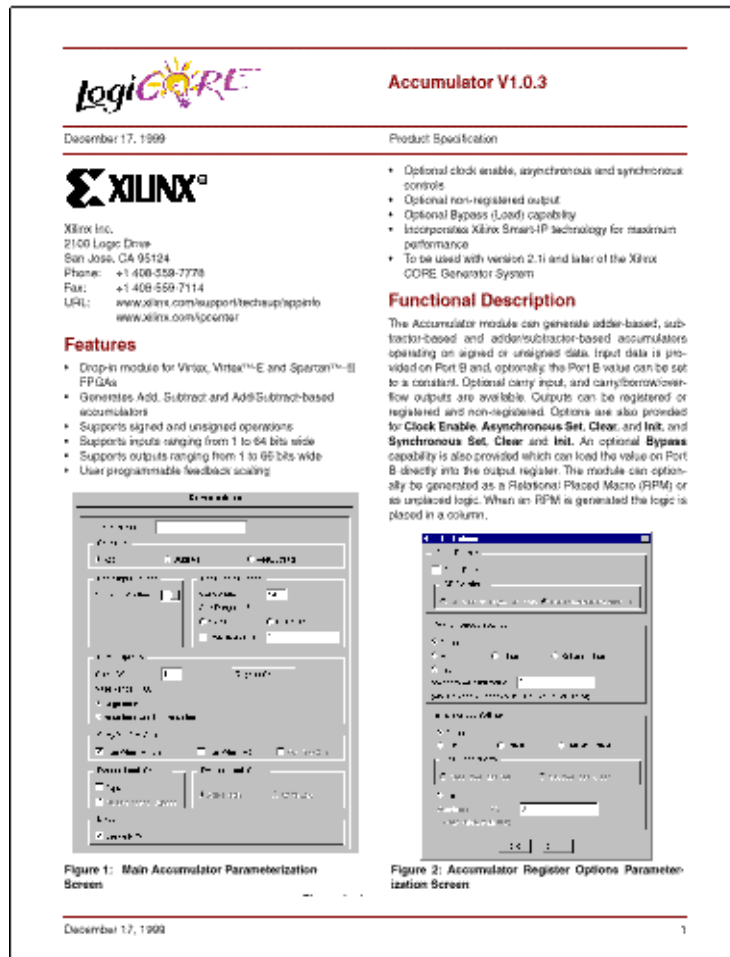


Figure 2-3 Data Sheets

The location of both the Web browser and the PDF Viewer can be set with the Preference Options dialog box, as shown in the following figure:

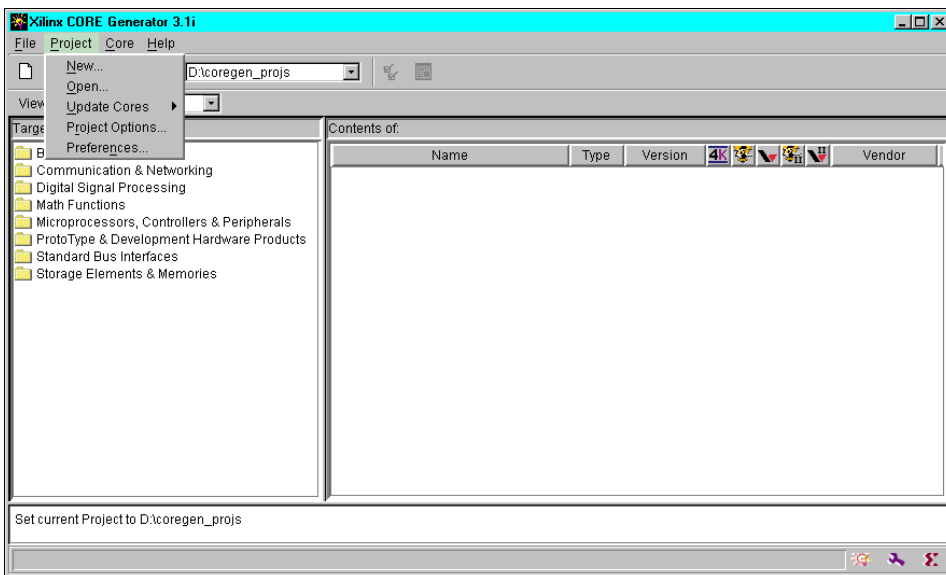


Figure 2-4 Preference Dialog Box

To locate the Web Browser and the PDF Viewer, use the following procedure:

1. Select **Project** → **Preferences**

The Preference Options dialog box appears.

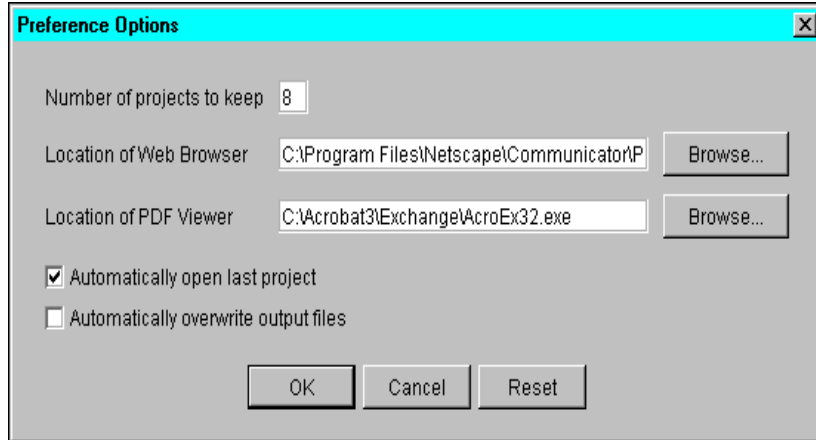


Figure 2-5 Web Browser and PDF Viewer

2. In the Location of Web Browser, browse to your default browser, for example, Microsoft Explorer.
3. In the Location of PDF Viewer, browse to the path of your PDF Viewer.

Sample Preference File

A sample workstation preference file follows:

```
#Coregen preferences
#Fri Apr 21 13:46:57 PDT 2000
LastProject=/home/myprojects/fir_filter
AlwaysOpenLastProject = false
OverwriteFiles = true
WebBrowser = /usr/bin/netscape
PDFViewer = /usr/local/bin/acroread
```

The registry entries are created using the Registry Editor, as shown in the following figure:

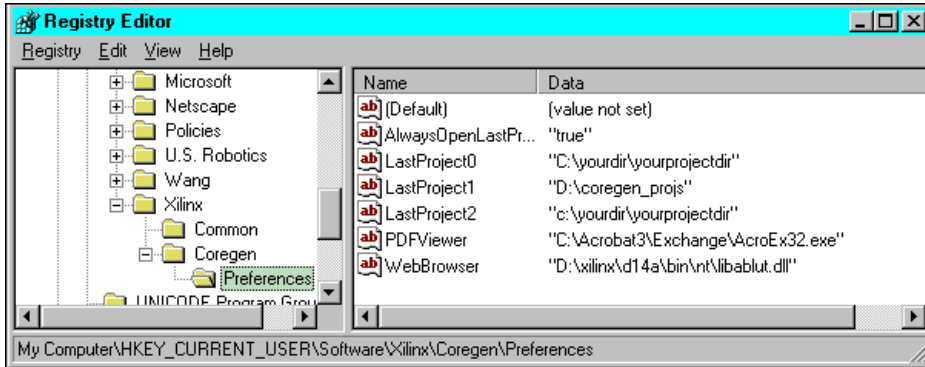


Figure 2-6 Registry Entries

The following table shows the list of supported preference file properties:

Table 2-1 Preference File Table

Field	Value	Description
Always OpenLastProject	False	Do not start CORE Generator System in the last project.
	True	Always start CORE Generator in the last active project.
LastNProjects	Integer value	Recalls number of previously opened projects (default=4).
Last Project	Integer value	
LastProject<n>	Integer value	Path to last project open by the user.
		Path to project n in list of previously open projects.

Table 2-1 Preference File Table

Field	Value	Description
OverwriteFiles	False True	Prompt user before overwriting files during elaboration. Automatically over-write design files during elaboration.
Browser	<path_to_web_browser>	Set fully qualified path to the user's web browser.
PDF Viewer	<path_to_pdf_viewer>	Set fully qualified path to the users' Acrobat, Netscape, or other PDF viewer.

Using the CORE Generator

This chapter explains the major functions performed by the designer when using the CORE Generator. The functions in this chapter include the following:

- “Using the CORE Browser”
- “Customizing a Core”
- “Specifying Command Files”
- “Generating Cores in Batch Mode”
- “Defining CORE Generator Command Line Options”
- “Updating Cores in the CORE Generator”
- “Integrating CORE Generator into Applications”
- “Listing Inputs and Outputs Files”

Using the CORE Browser

The main view of the CORE Generator is the Core Generator 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 of the Core Generator Browser allows you to browse through these folders. To select a folder, click once on the folder name in the left panel. To expand a folder, double-click the folder icon to the left of the folder name. The folder expands to reveal more folders. To close a folder, double-click the open folder icon. Some folders have a + icon or a – icon to their left. You can open or close the folder with a single click on the icon.

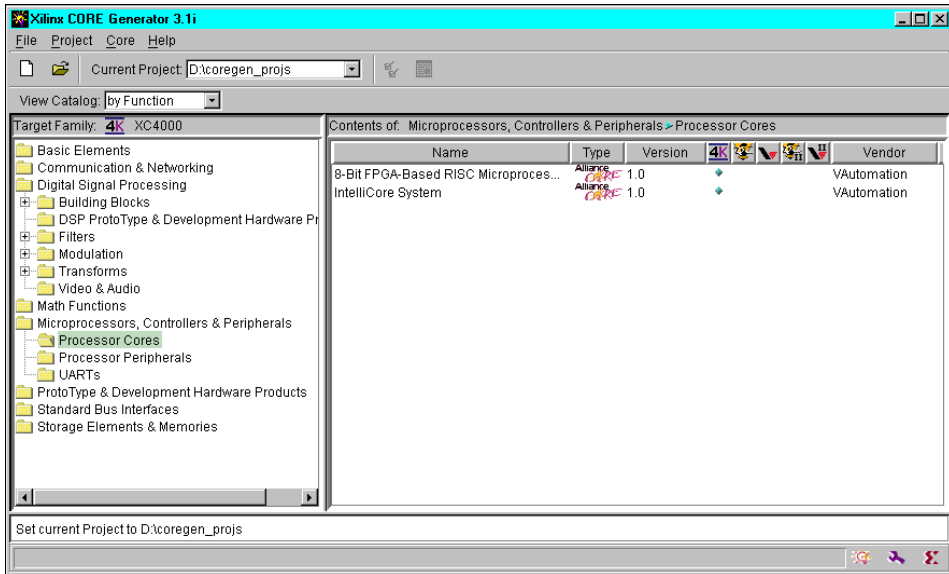


Figure 3-1 CORE Generator Browser

The cores in the selected folder are displayed in the right hand panel of the Core Browser. Cores are listed by name and also have type, version, family and vendor information displayed in columns. The size of each column can be altered by pointing the mouse at the separator between the column headings until the cursor changes to a  icon. Click and hold the left mouse button 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 size of each of the panels can be adjusted by pointing at the vertical or horizontal bar separating the panels until the cursor changes to a  icon (for vertical bars) or  icon (for horizontal bars). Clicking and holding the left mouse button allows the panels to be resized when the mouse is moved. Release the mouse button down when the desired panel layout is achieved.

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

Some cores in the right panel of the main window appear to be grayed out. This means that these cores are not available for the currently selected Xilinx FPGA family.

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

A core can be selected by clicking the name of the core in the right panel. After 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

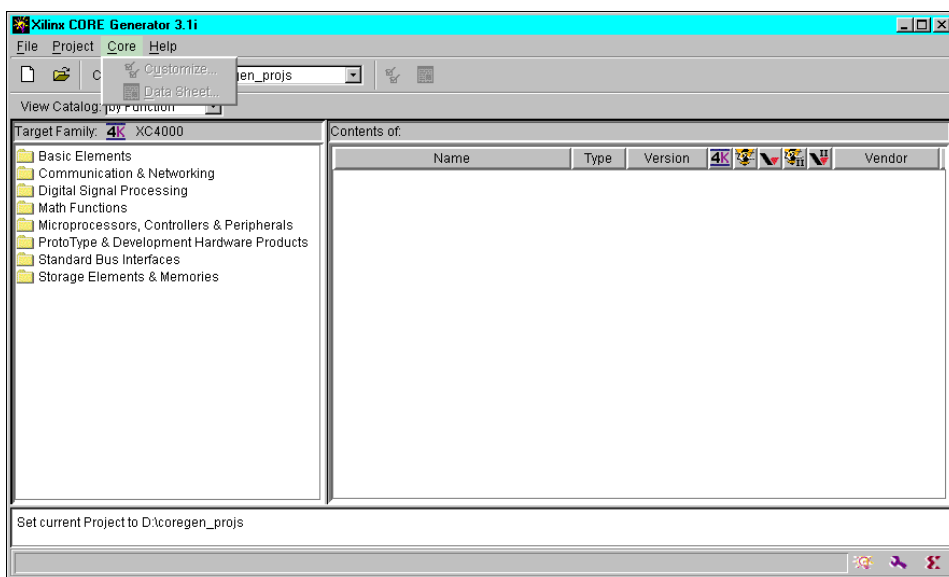


Figure 3-2 Data Sheet Selection Menu

Both of these actions launch the Acrobat Reader to display the data sheet.

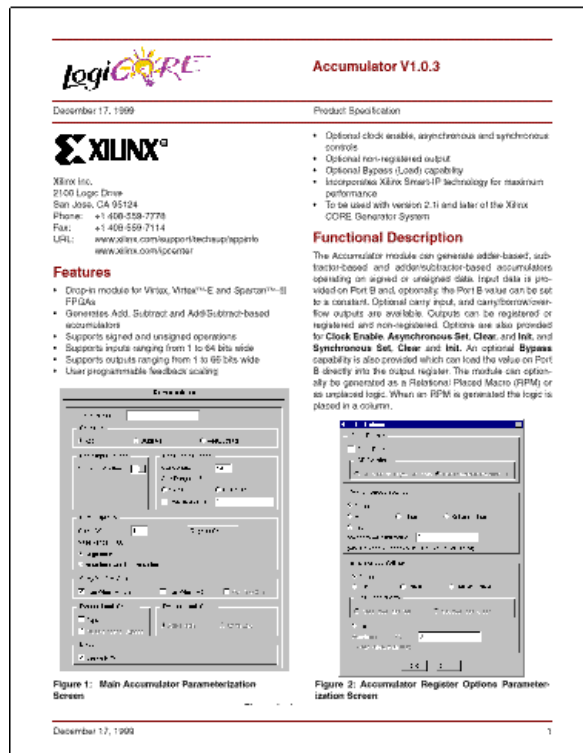


Figure 3-3 CORE Generator Data Sheet

You can also access a data sheet by pressing the right mouse button over a core in the right panel of the main window. Data sheets can be viewed for any core listed in the right panel whether they are grayed out or not.

Customizing a Core

Most cores have a customization GUI. To open a customization GUI, navigate to the module of your choice. From here, you can display the customization GUI for a core by proceeding with *one* of the following steps:

- Double-click the core in the right panel of the Core Browser, or
- Click on the Customize button on the Core Generator Browser toolbar, or

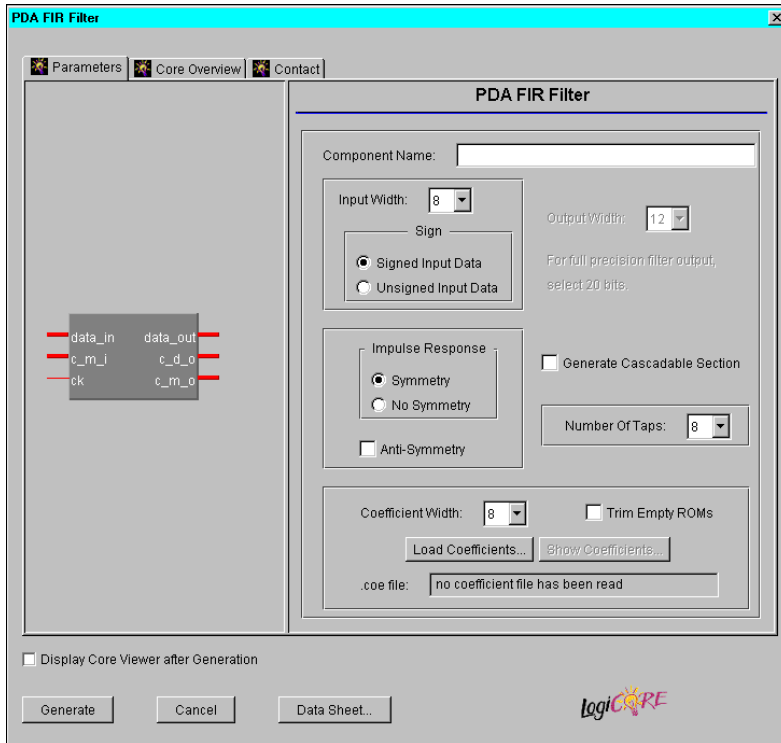
- Select the core in the left panel and select **Core** → **Customize**, or
- Press the right mouse button over a core in the right panel of the main window.

The customization GUI is only available for cores that support the currently selected Xilinx FPGA family.

While the customization GUIs are unique for each core, there are some characteristics that are common to all modules.

Displaying the CoreViewer

Selecting the CoreViewer checkbox brings up the CoreViewer after generation of the core is completed.

**Figure 3-4 CoreViewer Checkbox**

The CoreViewer shows a graphical representation of the cores footprint, which can be useful in floorplanning a large design. It also provides resource, and other statistical information about the implemented core as shown in the following figure:

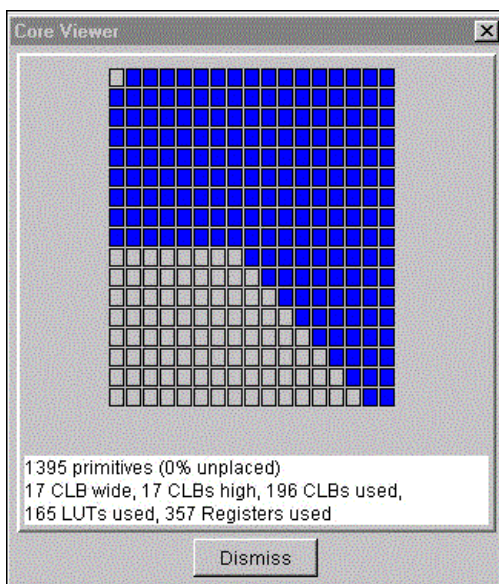


Figure 3-5 CoreViewer Screen

The default tab, that is displayed when the parameterization window is open, is the Parameter screen. A Core Overview screen is also available, as is a Contact Information screen. These tabs are accessed by clicking the appropriate tab at the top of the parameterization window.

Naming CORE Generator Modules

Most modules have a Component Name field which allows you to assign a name to the core that you create. Files that the CORE Generator creates for a particular core have a root filename that matches the Component Name. Component names have the following restrictions:

- Must begin with a lower case alphabetic character: a - z
- No uppercase letters
- May include (after the first character): 0 - 9, _(underscore)

- No extensions
- No HDL reserved words, for example,
Verilog: Do not use module, input, output
VHDL: Do not use component, function, configuration, port, signal

Using the Generate, Cancel, and Data Sheet Buttons

The Generate, Cancel and Data Sheet buttons are common to all parameterization windows. Assuming there are no conflicts with any of the specified parameters, pressing **Generate** causes the CORE Generator to create the requested files for the core. Pressing **Cancel** returns you to the Core Browser window without generating any files. Pressing **Data Sheet** invokes Adobe Acrobat to display the data sheet for the module being parameterized.

For information about a specific core's parameters, 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 way. The affected field is highlighted in red until the problem is corrected. If the reason 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, for example, PDA FIR, SDA FIR, RAM and ROM, and Virtex Block RAM, multiple coefficients or initialization values. To specify the values for these modules, you must load a .COE file using the Load Coefficient button in the parameterization window.

Module specific information about the requirements for a core's COE file can be found in that core's data sheet.

The following syntax displays the general form for a COE file:

```
Keyword =Value ; Optional Comment  
Keyword =Value ; Optional Comment  
CoefData =Data_Value, Data_Value,
```

The following table describes CORE keywords:

Table 3-1 Description of Core Keywords

Keyword	Description
CoefData	Used for filters to indicate that the data that follows comprises the coefficients of the filter.
MemData	Used for distributed memories.
MEMORY_INITIALIZATION_VECTOR	Used for Virtex Block memories.

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

The CoefData, MemData, and MEMORY_INITIALIZATION_VECTOR keywords must all be the last keywords in the COE file. Any keywords that follow them are ignored.

You can find examples of COE files for the PDA FIR, SDA FIR, distributed RAM, distributed ROM, and Virtex block RAM COE files in the \$XILINX/coregen/data directory. The following section displays examples of COE files:

```
***** 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;
```

Specifying Command Files

A Xilinx CORE Generator command file is a file that contains valid CORE Generator commands and comments. 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. You can also execute the command files 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 are as follows:

- coregen.ini / coregen_user_name.ini files
- User-generated command files
- XCO files
- coregen.log files

coregen.ini/coregen_user_name.ini

The CORE Generator supports the loading of INI files when they are first invoked and when changing projects. An INI file can contain any valid CORE Generator command. General preferences are stored on a per user basis and project options are stored with the project. In special situations, it is desirable to execute one or more commands on startup or when opening a project. When you first invoke the CORE Generator, it looks for a file named coregen.ini in the startup directory. Alternatively, you can direct the CORE Generator to read a specific INI file with `-i ini_file` on the command line. When opening a project, the CORE Generator looks for a coregen.ini in the project directory.

User-Generated Command Files

You can write your own command files to generate cores, create projects, customize the CORE Generator environment, or execute any other CORE Generator command. User-generated command files can 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 modify the current project. For more detailed information, please see the “Listing Inputs and Outputs Files” .

XCO Files

When generating a core, the CORE Generator creates a file called *component_name.xco*. This 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** or in batch mode. You can use an XCO file from a batch file to bring up the Customization GUI for a specific core with the values from the XCO. This is useful when a core needs to be regenerated but has parameter changes. The CORE Generator is run with the `-b` option pointing to this batch file.

Comment lines begin with the `#` character. Any output format or options lines start with the keyword `SET`. These options 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 customization GUI. All data read in from a `.COE` file is also preceded by the `CSET` keyword.

The following is an example for the FIR filter generated using the `pdafir.coe` file:

```
# 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

The corgen.log is a log file that is automatically written by the CORE Generator. The 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.

- (Windows) The corgen.log is written to \$XILINX\coregen\tmp.
- (UNIX Workstation) The corgen.log is written to the current project directory.

Generating Cores in Batch Mode

Running the CORE Generator System with no options selected causes the CORE Generator to start 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. The .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, then the CORE Generator System must be invoked using a fully specified path as follows:

```
coregen [ -i path_to_coregen_ini_file_name ] [ -p  
project_path ] [ -q polling_dir_path ] [-h] -b  
module_name.xco
```

Defining CORE Generator Command Line Options

The invocation of the CORE Generator System in batch mode is as follows:

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

The CORE Generator System's command line options are as follows:

-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. The *command_file_name* is the path to the command file to be executed.

-i coregen_ini_file_name

By default the CORE Generator System uses the profile that is in the specified project directory. If a different profile is required, then the path can be explicitly specified using a fully specified path name. The *coregen_ini_file_name* is the path to the CORE Generator INI file to be loaded. The profile also loads this file from the Project directory.

-p project_path

Specifies the project directory. The project path must be fully specified. The *project_path* is the path to the CORE Generator Project.

-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. The *polling_dir_path* is the polling directory.

-h

This option displays the CORE Generator batch mode help screen. The invocation of the CORE Generator system in batch mode is as follows:

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

Listing the CORE Generator Commands

The following table describes the CORE Generator commands / arguments and their functions:

Table 3-2 CORE Generator Commands

Command	Arguments	Function
CSET	<i>core_property>=value</i>	Sets a core property value.
END	N / A	Terminates CORE Generator session.
EXECUTE	N / A	Executes indicated command file.
GENERATE	N / A	Elaborates the currently selected core.
LAUNCH	N / A	Launches a core customization GUI.
NEWPROJECT	<i>project_path</i>	Creates a new project in the indicated directory. Not valid in a XCO command file.
SELECT	<i>core_name architecture vendor core_version</i>	Selects the indicated core.
SET	<i>global_property=value</i> <i>project_property=value</i>	Sets a CORE Generator property value.
SETPROJECT	<i>project_path</i>	Changes the current project to the indicated property value. Not valid in a XCO command file.

Listing the CORE Generator Global Properties

The following table lists the CORE Generator global properties, values, and their descriptions:

Table 3-3 Global Properties

Global Properties	Values	Description
CoreGenPath	<i>path</i>	Specifies the path to the CoreGen install directory
CoreSelect	SpecifiedVersion LatestVersion ProjectVersion	This property applies only when reading a XCO file in batch mode. Specified Version: Use only the core version specified. Latest Version: Use the latest version of the specified core available in any of the loaded libraries. Project Version: Use the version of the core specified in the project file.
DebugModes	true false (default)	Sets debug mode.
FoundationPath	<i>path</i>	Specifies the path to the Foundation install directory.
GuiMode	GuiOn GuiOff	Sets the GUI or batch modes.
LockProjectProps	true false	Set the global property to true to lock the project. To unlock, reset this property to false.

Table 3-3 Global Properties

Global Properties	Values	Description
ProjectOverride	true false	Specify true to use the current project's attributes instead of those listed in the XCO file. Specify false to use those attributes in the XCO files.
Username	<i>username</i>	This is your login name.

Listing Project Properties

The following table describes the Project Properties commands, values, and their functions:

Table 3-4 Project Properties

Command	Values	Description
BusFormat	BusFormatAngleBracket BusFormatSquareBracket BusFormatParen BusFormatNoDelimiter	Sets the indicated output bus formatting.
DesignFlow	Schematic VHDL Verilog	Schematic generates EDIF and VHDL. Verilog generates the corresponding HDL configuration files to be used with other static behavioral HDLs provided by CORE Generator
ExpandedProjectPath	<i>project_path</i>	Specifies the expanded path to the CORE Generator install directory.

Table 3-4 Project Properties

Command	Values	Description
FlowVendor	Foundation Viewlogic Mentor Cadence Synplicity Synopsys Exemplar Other	Signifies which vendor toolset you have selected to simulate and develop the core design.
ProjectPath	<i>project_path</i>	Specifies the path to the current project.
SimulationOutputProducts	VHDL Verilog	Specifies the outputs products for elaboration.
ViewlogicLibraryAlias	ViewlogicLibraryAlias / <alias_name>	Specifies the Viewlogic library alias. Defaults to <i>primary</i> on UNIX.
XilinxFamily	XC4000 Spartan Virtex Spartan2 Virtex2	Consists of targeted architecture.

Updating Cores in the CORE Generator

The CORE Generator is capable of handling multiple versions of any core. The ability to create new versions of a core, while maintaining existing versions allows a designer to introduce additional functionality. A designer can also fix problems in an earlier version of a core without forcing all users of a core to use the new version.

Each project maintains a list of cores visible to the project and their version number. This list is located in the section on the right of the main CORE Generator window. Only one version of each core is visible at any one time. All cores in the repository are available through batch mode so an older core can be regenerated at any time. When a new project is created, the latest version of all the cores is added to the project. You can customize a project to change the specific versions visible within a project.

Downloading New Cores

When new cores and new versions of existing cores are downloaded from the IP Center, they are installed in the CORE Generator's hierarchy but are not visible to existing projects. This capability exists to insulate existing projects from updates to the cores used in that project. Any changes in the functionality does not impact existing projects since new cores are not automatically updated for existing projects. The multiple version support capability exists to allow a new core or new version of an existing core to be made available in an existing project.

The new cores and their installation instructions can be downloaded from the Xilinx IP Center at

<http://www.xilinx.com>

Updating a Core Version in an Existing Project

When new cores and new versions of existing cores are downloaded from the IP Center, they are added to the repository but are not automatically available in existing projects. After new cores have been added and a project is opened, a dialog box is shown asking you if you want to update the project list of visible cores. This capability exists to insulate existing projects from unwanted changes while still allowing you to update projects easily.

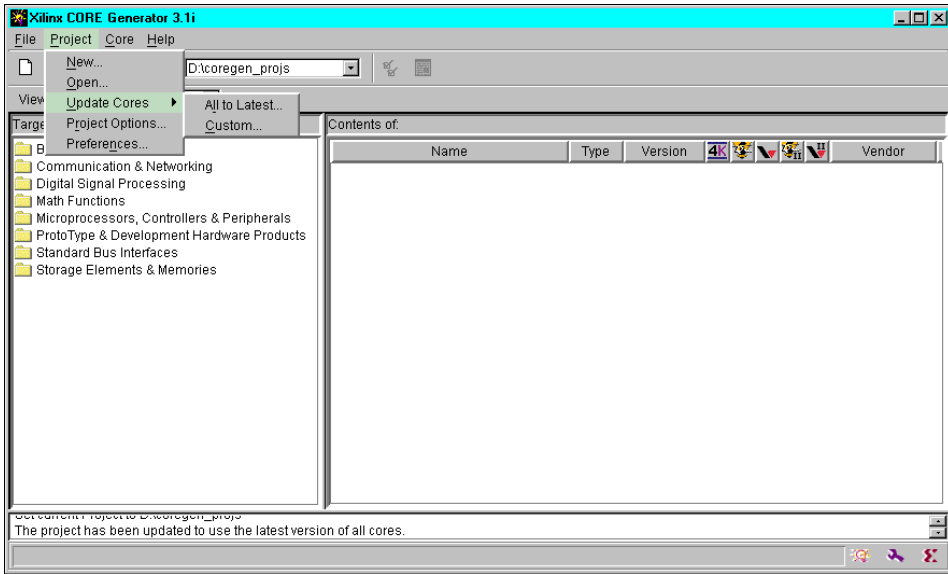


Figure 3-6 Update Dialog Menu

All to Latest

Updates all the cores in the project to the latest version available in the repository and adds all new cores.

Custom

Launches the Custom Update Window and allows you to selectively add new cores and /or versions of cores.

Understanding the Update Project Cores Menu

The procedure for updating existing cores to the newest cores version follows:

1. Select **Project** → **Update Cores** → **All to Latest**.
This menu selection can be used at any time to update all the cores in the project to the latest version available in the repository and add any new cores.

2. Select **Project** → **Update Cores** → **Custom**.
Use this selection to customize your cores by adding or removing individual cores to the current project profile.
3. The *Update Project Cores* screen appears and offers you the following folder selections, for example, catalog selections by function:

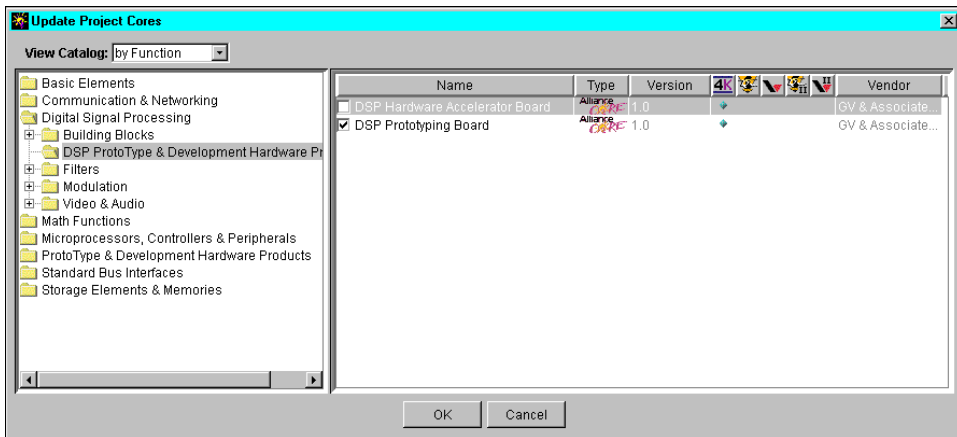


Figure 3-7 Update Project Cores Custom Window

- ◆ Basic Elements
- ◆ Communication & Networking
- ◆ Digital Signal Processing
- ◆ Math Functions
- ◆ Microprocessors, Controllers & Peripherals
- ◆ ProtoType & Development Hardware Products
- ◆ Storage Elements & Memories

Removing Cores

Use the following procedure to remove individual cores to the current project profile:

1. Select **Digital Signal Processing** → **DSP ProtoType & Development Hardware Products** as shown in the following figure:

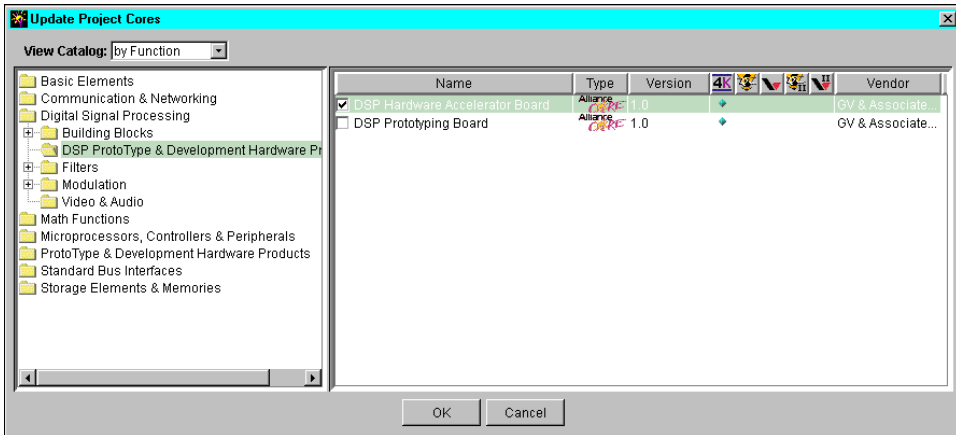


Figure 3-8 Update Project Cores Custom Window

2. Select **DSP Hardware Accelerator Board**
3. The core is removed.

Using the get_models Command

The `get_models` program is a command line utility that is used to manually extract the Verilog or VHDL behavioral models embedded within a user's CORE Generator System installation to a single, central location. In the 3.1i and later releases of the CORE Generator System, `get_models` is run automatically during the installation of the CORE Generator software. You do not need to run the `get_models` since it is also run during installation of post 3.1i CORE IP updates. You can run the `get_models` manually if for example, the install extraction fails.

In a Xilinx software installation, CORE Generator models may exist in the CORE Generator tree (\$XILINX/coregen) in either archived or source file format. For Verilog interpretive simulators such as Cadence Verilog-XL, extracting the behavioral models to a single

library directory collects them together in one location so that they can be referenced from a common location by the simulator. For compiled simulators (all VHDL simulators, and some Verilog simulators such as Synopsys VCS), extracting the behavioral models to a single library directory allows them to be conveniently analyzed by your Verilog or VHDL simulator.

You can use the `get_models` program to extract individual behavioral models for specific CORE Generator modules to your project directory if preferred.

Syntax

```
get_models [-h | -help] [-vhdl] [verilog]  
[-dest destination directory]
```

Required Parameters

```
-verilog | -vhdl
```

Using these parameters extracts Verilog or VHDL behavioral models only. Specify either the `-verilog` or `-vhdl` options when running `get_models`.

Note When specifying the `-verilog` option, specify the explicit path to the destination directory to tailor the 'include statement in the .VEO template file correctly.

Optional Parameters

```
-dest destination_directory
```

Use this parameter as the target location when creating the `xilinx-corelib` and any `VendorCoreLib` subdirectories. The metacharacters `.` and `..` are supported. The `xilinxcorelib` subdirectory will always be one of the directories created in this directory. The Verilog default location of `destination_directory` is `$XILINX/verilog/src`. The VHDL default location of `destination_directory` is `$XILINX/vhdl/src`. For networked UNIX workstations, you may need system administrator privileges to extract models to this Xilinx software directory location.

Inputs

The inputs to the `get_models` utility are the CORE Generator behavioral models located in your Xilinx CORE Generator System installation. The models exist in either of the following two formats:

- Archived together with other data files associated with a given IP module, for example, JAR (JAVA Archive format) file, located at `$XILINX/coregen/ip/com/xilinx`.
- Non-archived source file format, .V and .VHD files, in a simulation subdirectory of a given Core Generator IP module directory. CORE Generator IP module data files are located at `$XILINX/coregen/ip/com/xilinx`.

Note You may not need to specify the path to these inputs when using the `get_models` utility. The path to the behavioral models is implied by the value of your XILINX environment variable.

Outputs

The `get_models` utility produces a directory of extracted source format Verilog or VHDL behavioral models named `xilinxcorelib` or `VendorCoreLib`. In addition, an ASCII text file containing a comprehensive listing of the extracted models is created. For Verilog compiled simulators, the order indicated in the `verilog_analyze_order` file is only a suggested order, since Verilog simulators do not require models to be compiled strictly in a top-down order.

In contrast, VHDL simulators require models to be compiled from the bottom-up using lower level blocks before higher level blocks, and there can be more than one compile order that meets this bottom-up order requirement. For instance, the order specified in the `vhdl_analyze_order` file is an example of one such combination that the `get_models` option generates automatically for you. The `get_models` also generates a `get_models.log` log file in the destination directory.

Integrating CORE Generator into Applications

The CORE Generator provides a number of interfaces for integration into other applications. The Xilinx Foundation series of tools is a good example of the level of integration that is possible. The Xilinx Alliance series provides integration with various third party CAE tools. This section describes the specific interfaces provided by the CORE Generator that allow others to integrate it into their applications.

```
SETPROJECT project_name
```

```
LAUNCHXCO sco_filename
```

You can then run CORE Generator with the `-b batch_filename` command.

Polling Mode

The CORE Generator can be invoked in the polling mode, which looks the same as the standard GUI mode. Polling mode allows an application to communicate to CORE Generator through files. This mode is useful to an application that needs to run CORE Generator continuously in the background while frequently checking to see if CORE Generator has generated a core, and occasionally issuing instructions to the CORE Generator. CORE Generator can read and write files in polling mode and the application uses the `-q` option of *poll_dir_path* to specify where the files are located.

Output Polling Files

Output polling files are written by CORE Generator to communicate to an application when CORE Generator has finished generating a core. Output polling files always have the name `coregen.fin` and they contain two lines.

1. The first line contains the following:
 - ◆ User assigned core name
 - ◆ Name of the core
 - ◆ Core version
2. The second line contains the keyword `SUCCESS` or `ERROR` depending on whether the core was successfully generated. For example,

```
coregen.fm
```

```
regaddr Registered_Addr 1.0
```

```
SUCCESS
```

When an application finds the keywords, `SUCCESS` or `ERROR` in the `coregen.fin` file, the application reviews the various log files to determine the appropriate processing for the core. See the “Listing Inputs and Outputs Files” section for details on log files. The application should delete the `coregen.fin` file immediately after it has been processed so that the CORE Generator is free to write a new file. The CORE Generator overwrites `coregen.fin` on the next core generation.

Input Polling Files

The CORE Generator uses the input polling file to receive commands from the invoking application. The input polling file always has the name `CORE Generator.com`. CORE Generator can contain any number of valid CORE Generator commands and is terminated by the keyword, `COMPLETE`. While in the polling mode, CORE Generator frequently monitors the status of the `CORE Generator.com` file. When it detects the keyword, `COMPLETE`, CORE Generator sequentially executes the commands in the file. CORE Generator deletes the `CORE Generator.com` file after completion of the last command.

ASY and XSF Files

The ASY file is an ASCII file containing graphical symbol information and pin attributes. The ECS schematic editor uses this file to generate symbols.

The XSF pin file is used by the Foundation tools to create a symbol representing the core. The ASY graphical symbol file is used by the Foundation iSE tools to display a symbol representing the core.

Listing Inputs and Outputs Files

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

Table 3-5 CORE Generator Input Files

File Extension	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 for regenerating a core. It can also be used as a logfile to determine the settings used to generate a particular core. This file is 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 “XCO Files” section of this chapter.

Table 3-6 CORE Generator Output Files

File Extension	Description
.ASY	Graphical symbol file. Used by the Foundation iSE tools to create a symbol representing the core.
.EDN	EDIF Implementation Netlist for the core. Describes how the core is to be implemented. Used as input to the Xilinx Implementation Tools.
get_models.log	Log file containing all user visible messages displayed during a get_models run. The log file is written to the get_models destination directory.
.MIF	Memory Initialization File which is automatically generated by the CORE Generator System for Virtex Block RAM modules when an HDL simulation flow is specified. A MIF data file is used to support HDL functional simulation of Block RAM modules as well as to specify the many instantiation values for the implementation netlist.
.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 "Using the CORE Generator Verilog Design Flow Procedure" in Chapter 4.

Table 3-6 CORE Generator Output Files

File Extension	Description
verilog_analyze_order	This file lists the CORE Generator Verilog behavioral models in a suggested compiled order before performing a behavioral simulation in a compiled simulator. This applies to compiled Verilog simulators only, for example, Synopsys VCS and Cadence NC-Verilog.
.VHO	VHDL Template file. The components in this file can be used to instantiate a core. For more details, refer to the section on “Using the CORE Generator VHDL Design Flow Procedure” in Chapter 4.
vhdl_analyze_order	This file lists the CORE Generator VHDL behavioral models in the order that they must be compiled for simulation. More than one compile order may be valid for the library.
vlink.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 functional simulation.
.XSF	A pin file used by the Foundation tools to create a symbol representing the core.
XilinxCoreLib/*.v	Verilog behavioral models extracted from the IP installed in the CORE Generator tree.

Table 3-6 CORE Generator Output Files

File Extension	Description
XilinxCoreLib/*.vhd	VHDL behavioral models extracted from the IP installed in the CORE Generator tree.
XilinxCoreLib/*_comp.vhd	VHDL component declaration files for each CORE Generator IP module extracted from the CORE Generator.

Understanding CORE Generator Design Flows

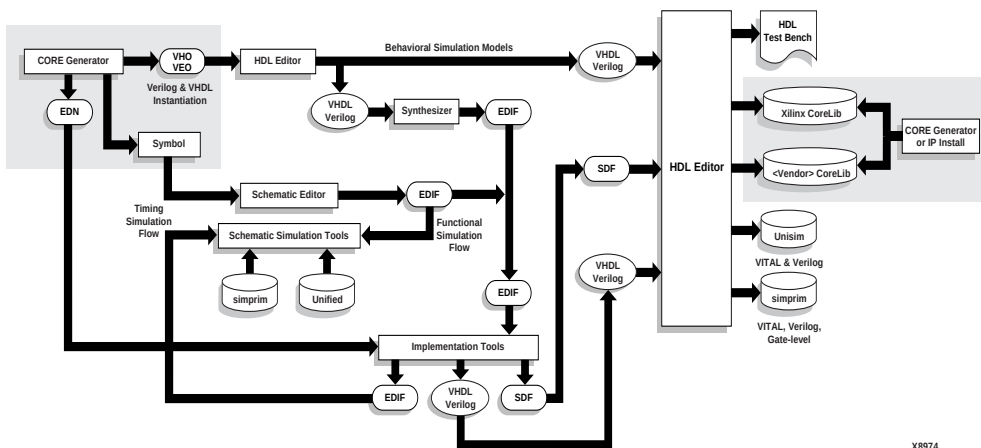
This chapter describes how to integrate a CORE Generator module into a user design through the use of various design flows; schematic and HDL. This chapter contains a general overview of the design flows, which include the following topics:

- “Understanding CORE Generator Design Flow Basics”
- “Describing the CORE Generator Schematic Design Flow”
- “Describing the HDL Behavioral Model Delivery System Features”
- “Using the CORE Generator Verilog Design Flow Procedure”
- “Using the CORE Generator VHDL Design Flow Procedure”

Understanding CORE Generator Design Flow Basics

The CORE Generator System produces a Electronic Data Interchange Format (EDIF) netlist, and may also produce a Foundation or Viewlogic schematic symbol, and /or a Verilog Output (VEO) or VHDL Output (VHO) template file. The Electronic Data Netlist (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 Electronic Design Automation (EDA) tools. Finally, the VEO and VHO template files contain code that can be used as a model to instantiate a CORE Generator module in a Verilog or VHDL design so that it can be simulated and integrated into a design.

The grayed areas in Figure 4-1 indicate the portions of the design flow directly associated with the CORE Generator. The left-side gray area shows the EDN, VEO, VHO, and schematic symbol files produced by the CORE Generator System. The right-side gray area shows the XilinxCoreLib and VendorCoreLib source libraries that are created or updated during CORE Generator and IP module update installation. These libraries contain the behavioral simulation models for the CORE Generator cores.



X8974

Figure 4-1 CORE Generator Flow Chart

Describing the CORE Generator Schematic Design Flow

The CORE Generator System produces an EDIF Implementation Netlist (EDN) for schematic design flows. For Viewlogic and Foundation flows, the CORE Generator also produces a schematic symbol. The EDN file contains information for implementing the module. The Foundation symbol or Viewlogic symbol allows you to integrate the module into a schematic for these EDA tools.

Starting a Schematic Design Flow with Viewlogic

The Viewlogic interface outputs are generated by a script, `vlink`, which calls for both Viewlogic and Xilinx tools. You need to setup your platform with the following instructions:

- (Windows) Install both the Viewlogic and Xilinx Implementation Tools software in order to generate the outputs required to integrate a core into a Viewlogic design.
- (UNIX workstation) Set your environment to run both the Viewlogic tools and the Xilinx Implementation software. If you run Powerview v6.1 you need to have your environment setup to run Viewlogic Fusion v1.4 or later. This setup gives you access to the VHDL2SYM utility for symbol generation. If any of these tools are not installed or not properly set up in your environment, then diagnostic errors similar to the following example may be reported in the CORE Generator console, `coregen.log`, or `vlink.log` file in your project directory:

```
"WARNING: Core xxx did not generate product  
ViewSym"
```

For both platforms, you need to place your CORE Generator project in a valid Viewlogic project directory. A valid Viewlogic project directory consists of a project defined in a local project `viewdraw.ini` file. This file is automatically created in Workview Office by the Workview Office Project Manager.

Note For details on Viewlogic and Xilinx Implementation Tools setup, please refer to the *Alliance or Foundation Quick Start Guide*.

Creating a Viewlogic Project

The following procedure describes the process for creating a Viewlogic project:

1. Create a directory for a Viewlogic project by entering the following path:
`c:\wvoffice\project`
2. Set up project libraries
 - ◆ (UNIX Workstation) Define the project libraries in the `viewdraw.ini` file located in the project's working directory.
 - ◆ (Windows) Set up the project libraries through the Project Manager GUI.

Continue with the following steps:

3. Open a New project.
4. 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.
5. Add the project directory as a Writable library and give it the alias, *primary*. Move it to the top of the library search order.
6. 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 (logi-
blox)
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 looks 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.

7. Set the Output Format. From the CORE Generator Project menu, select Project Options to open the dialog box shown in the following figure:

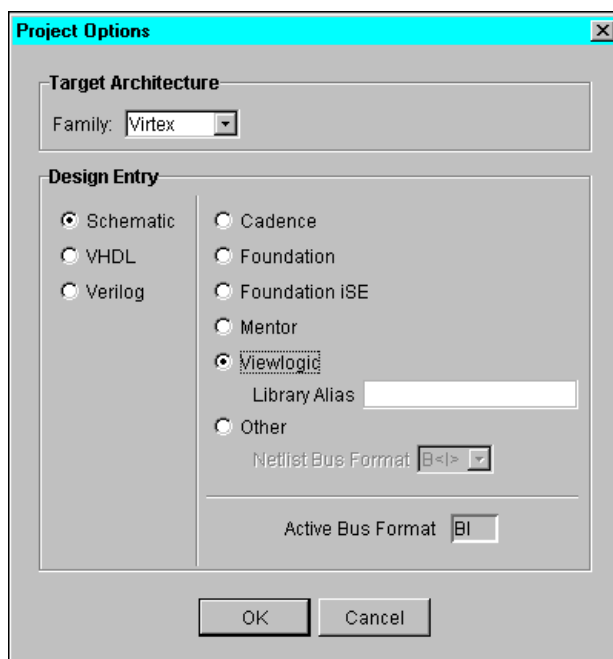


Figure 4-2 Project Options Dialog Box

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.

8. Set Project Path and Viewlogic Library alias from the CORE Generator Project menu.

Select **Project** → **Open**. Set the Project Path to point to the Viewlogic project directory you are working on, for example, `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 you can use any name with up to 8 characters.

Creating Output Files

Use the following procedure to create a Viewlogic symbol, a Viewlogic simulation file, and a netlist file.

1. Select Desired Module

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

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

This creates a Viewlogic Symbol, a Viewlogic WIR file, and a netlist file (.EDN). 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.

The simulation file is created from the EDN file and placed in the WIR subdirectory within the Viewlogic project. The .EDN file, used for implementation, is placed directly in the Viewlogic project directory.

The WIR simulation file is created from the EDN file and placed in the WIR subdirectory within the Viewlogic project. The WIR file is used by Viewlogic to generate the .VSM file for functional simulation and should not be deleted. In order to generate this file the CORE Generator System needs to access several Xilinx implementation tools executables. WIR file generation may fail if your environment is not set up properly. If an error occurs during the generation of these files, check the *vllink.log* file located in the Project Directory.

Note Workview Office 7.4/7.5, and Powerview 6.0 partially support Virtex and Spartan2 simulation in Viewsim. Use either VHDL or Verilog to simulate Virtex designs created with Viewlogic schematics. Please refer to Xilinx Solution 4318 at the following location for the latest information in Viewsim simulation support for Virtex:

<http://www.xilinx.com/techdocs/4318.htm>

4. Load the Symbol in the 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 to the schematic.

5. Attach a LEVEL property to the symbol with a value of XILINX. Check that this property has been added correctly to the symbol by displaying all properties attached to the symbol.
6. Connect the symbol to the rest of your design.
7. Check and save your schematic design.

When executing the Viewlogic Check program, error messages like the following example, are displayed for every CORE Generator module but can be safely ignored.

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

These error messages can be safely ignored. They are the by product of generating the Viewlogic symbol with a *Composite* block type to allow generation of the WIR files.

From now on, use the same flow for processing this design as you would for using macros from the Unified Library. For further information, refer to the *Viewlogic Interface/Tutorial Guide*.

Foundation Design Flow

Beginning with the 2.1i release of the Xilinx Foundation Series toolset, the CORE Generator System is integrated into the Foundation Project Manager, the Schematic Editor, and the HDL Editor. Please refer to the 3.1i documentation for details on integrating your CORE Generator module into a Foundation schematic design or HDL design.

Foundation ISE Design Flow

For details on how to integrate CORE Generator modules into a Foundation ISE design, please refer to the *Foundation Series ISE Guide*.

Mentor Design Flow

Beginning with the 2.1i release, the CORE Generator System is integrated into the Mentor Design Architecture. Please refer to the *Mentor Interface Guide* 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>.

Describing the HDL Behavioral Model Delivery System Features

The HDL behavioral model delivery system features a parameterized behavioral simulation model library (XilinxCoreLib), a instantiation template file, and a optional unused pin support that minimizes reliance on the Xilinx Mapper for removal of extraneous logic.

XilinxCoreLib Simulation Library

The CORE Generator System provides both Verilog and VHDL XilinxCoreLib behavioral simulation libraries to support the CORE Generator cores. The Verilog library is located at `$XILINX/verilog/src/XilinxCoreLib`, and the VHDL library is located at `$XILINX/vhdl/src/XilinxCoreLib`. In the previous 2.1i release, the libraries were extracted manually with the `get_models` utility. Starting with the 3.1i release, the XilinxCoreLib libraries are provided in source file format at standard locations in the Xilinx installation tree.

Note The libraries are now automatically updated by the IP Update installer when installing Core IP updates.

coredb

The `coredb` command line utility updates or regenerates the `coredb.xml` installed IP database file. The processes executed by `coredb` to update the `coredb.xml` database run automatically during the first CORE Generator GUI startup session, after installation of an IP module update. You can also run the `coredb` manually to explicitly force an update of `coredb.xml`.

Syntax

`coredb`

Instantiation Template Files

The `.V` and the `.VHD` behavioral models for each CORE Generator module are not copied to your project directory. The CORE Generator System only writes out `.VEO` and `.VHO` HDL instantiation template files. These instantiation template files contain pointers to the generic, parameterized HDL simulation models in the XilinxCoreLib libraries. In the case of a hierarchical behavioral model, pointers to the lower level behavioral models referenced by a higher level behavioral model are also written to the instantiation template files.

Support for Unused Optional Pins

The parameterized Xilinx CORE Generator HDL behavioral models are written to support unused optional pins on CORE Generator modules. The models are written to support those users who might not require all the input and output ports on a CORE Generator, for example, clock enable, and registered versions of outputs. Optional pins are omitted from the EDIF implementation netlist for the core when the core is generated if their associated functions are not requested by the user. The result is less reliance on the Xilinx Mapper to remove extraneous or unused logic.

verilog_analyze_order File

This file lists the CORE Generator Verilog behavioral models in the order in which it is suggested that they be compiled before performing a behavioral simulation in a compiled simulator. This applies to compiled Verilog simulators only such as Synopsys VCS, and Cadence NC-Verilog.

vhdl_analyze_order File

This file lists the CORE Generator VHDL behavioral models in the order in which they must be compiled for simulation. More than one compile order may be valid for the library.

Using the CORE Generator Verilog Design Flow Procedure

This section briefly describes the procedure for behavioral simulation, synthesis, and implementation of Verilog designs containing CORE Generator modules. Please see “Understanding the Verilog HDL Design Flow” in Chapter 5 for more details.

1. Generate the module.

Specify the Target Architecture and Verilog Design Entry settings for the project. Select the desired module, specify the customization parameters for the module, then generate the module.

2. Instantiate the module in the parent design.

Insert the instantiation template from the .VEO files for each module into the parent design, and edit the module connections.

3. Create *module.name.v* from the .VEO file.

Copy the .VEO file to *module_name.v*. Edit the ``include` statement from the .VEO file for each user generated module to reflect the path to your XilinxCoreLib directory. Comment out the instantiation template section in *module_name.v*.

4. Create a testbench.

Perform the behavioral simulation, including *module_name.v* in the command line.

5. Perform the behavioral simulation, including *module_name.v* in the command line.

6. Synthesize the design, applying any required black-box properties to the CORE Generator modules.

7. Write out the implementation netlist.

8. Implement the design using the Xilinx tools.

Using Instantiation Templates

Instantiation template files are files containing code that can be used to instantiate your CORE Generator module into your Verilog or VHDL design, and also contain code that supports behavioral simulation. The CORE Generator .VEO instantiation template file is automatically generated when you select Verilog as one of your Design Entry options in the Project Options menu. For the VHDL version, please see the “Verilog Instantiation Template for an 8-Bit Adder” section.

Using a .VEO Instantiation Template File

A Verilog instantiation template consists of the following components:

- Simulation Model include statement
- Instantiation code
- Module port declaration with customization parameters, as shown in the following example:

Verilog Instantiation Template for an 8-Bit Adder

```
/*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
```

Using the CORE Generator VHDL Design Flow Procedure

This next section consists of a brief description of the VHDL Design Flow procedure using the MTI ModelSIM. Please see Chapter 5 for more details.

1. Generate the module. Specify the Target Architecture and VHDL Design Entry settings for the project.
2. Analyze the xilinxcorelib library models as follows:
 - ◆ Create a library for the analyzed behavioral models named xilinxcorelib, using MTI ModelSIM.
 - ◆ Establish a link to the location where the analyzed behavioral models will reside using MTI ModelSIM.
 - ◆ Analyze the source behavioral models in `$XILINX/vhdl/src/xilinxcorelib` to the newly created library.
3. Instantiate the module in the parent design.
 - ◆ Copy the component, instance, and configuration declarations into the parent design.
 - ◆ Edit the instantiation template to connect the core to the parent design.
4. Simulate the design.

Analyze the parent design and testbenchfile. Simulate the testbench VHDL CONFIGURATION.
5. Synthesize the design.
6. Write out the implementation netlist for the design.
7. Implement the design using the Xilinx tools.

Using a .VHO Instantiation Template File

A .VHO file contains code that instantiates your CORE Generator module into your VHDL design. It 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 section 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 this release.

A VHDL instantiation template consists of the following components:

- xilinxcorelib LIBRARY declaration
- COMPONENT declaration
- Instantiation template
- CONFIGURATION declaration with VHDL generics, as shown in the following example:

VHDL Instantiation Template 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
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
```

The lines between the two markers –CONF_TAG and CONF_TAG_END must be added to a CONFIGURATION statement in your VHDL test fixture file or top level design file, for example,

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

Understanding the HDL Design Flow

This chapter describes the elements in a Hardware Description Language (HDL) design flow in the CORE Generator System environment and contains the following sections:

- “Using the HDL Behavioral Model Delivery System”
- “Understanding the Verilog HDL Design Flow”
- “Understanding the VHDL HDL Design Flow”

Using the HDL Behavioral Model Delivery System

To integrate a CORE Generator module into a HDL design, you need to start with the following procedure:

- Generate the module
- Instantiate the module in your design
- Perform a behavioral simulation of your design with the integrated module
- Analyze the models
- Simulate the models
- Synthesize the design
- Implement the design

Understanding the Verilog HDL Design Flow

This section describes the procedure for behavioral simulation, synthesis, and implementation of Verilog designs containing CORE Generator modules using the following third party vendor tools:

Synthesis:

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

Simulation

- MTI ModelSim / VLOG
- Cadence Verilog-XL

Describing the Verilog Design Flow Procedure

This next section describes in detail, how to use create a Verilog design.

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

To generate the module, start the Xilinx CORE Generator by selecting **Project** → **New Project**.

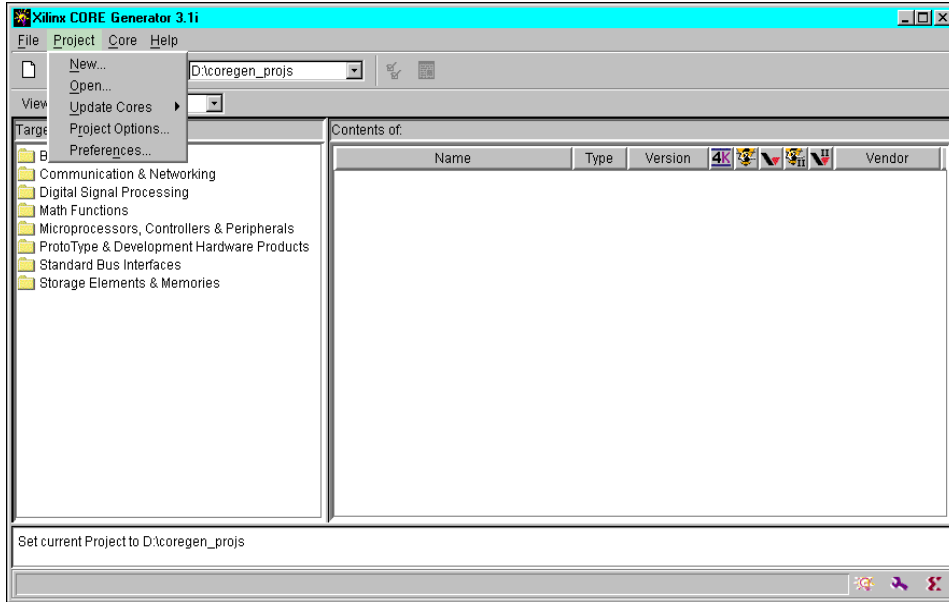


Figure 5-1 New Project Menu Screen

Select **Project Options** → **Design Flow** → **Verilog** → **Vendor**

This specifies the synthesis vendor software to synthesize your Verilog design. The EDIF bus delimiter format required for this flow is written out by the synthesizer for your design. This ensures for proper integration with the upper level parent implementation netlist.

In Figure 5-2, the Design Entry is set to Verilog, and the Vendor is set to Synopsys. As a result, the bus delimiter format is automatically set to B<I>

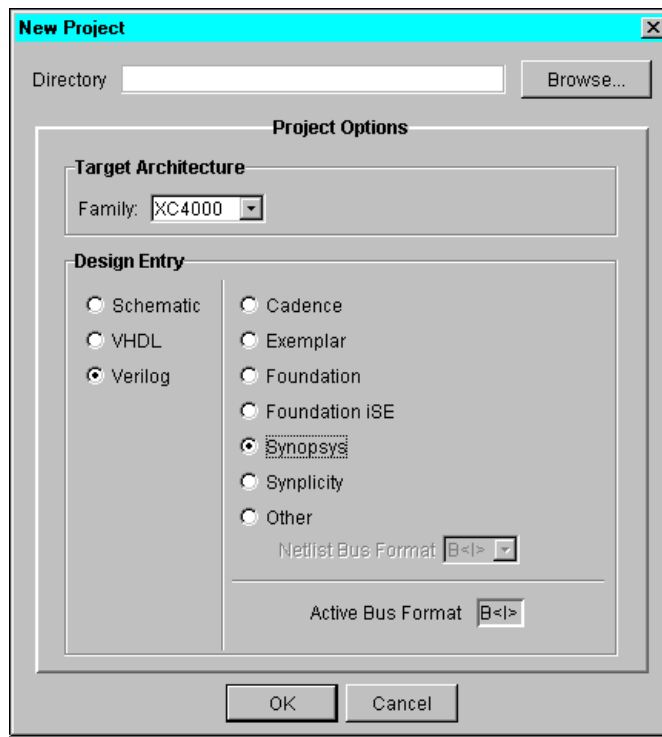


Figure 5-2 Verilog Behavioral Simulation Option

2. Prepare for simulation by extracting the behavioral models to a source library.module.

Install the CORE Generator in the \$XILINX/coregen directory.

To simulate cores, you need to extract the behavioral models from the CORE Generator system installation area to a directory, referred to as <destination_directory>. The recommended location of <destination_directory> is \$XILINX/verilog/src. This is the location of the source code for all other Xilinx Verilog libraries.

Extract the CORE Generator Verilog Behavioral Models to a source library. This step is required for all simulators.

3. Instantiate the module in the parent.

Insert the instantiation template from the .VEO files into the parent design, and edit the module connections. Create *module_name.v*. Edit the 'include library inclusion section to reflect the actual location of the extracted behavioral model source library. Comment out the instantiation template section in the *module_name.v*.

Select **Design Entry**→**Verilog**

Both a Verilog template file *component_name.VEO*, and an implementation netlist *component_name.EDN* are generated whenever a core is generated. The .VEO template file includes the following items:

- ◆ Library `include statement
- ◆ Module declaration section
- ◆ Module instantiation template

The following example illustrates the use of the Verilog template file with a parent design. Copy the module instantiation template and paste it into the parent design, as described in the following section:

Code Example 1 Using the Verilog myadder8.veo Instantiation Template File

```

/*****
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 editing the ‘include’ statement to reflect the actual location of the extracted xilinx-corelib and / or *Vendor-CoreLib* libraries 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));
*/
```

Analyze the resulting myadder8.v file, along with the parent design when preparing for simulation.

Using an Alternate Method for Instantiating Modules

You can also use an alternate method by pasting the library declaration, module declaration and instantiation template for the core into the parent Verilog design file. Begin the alternate pasting method, paste the VERILOG parent design file: myadder8_top.v. The component, myadder8, is instantiated, and the module is declared. The instantiation template is copied from myadder8.veo, and pasted into the parent design.

```
myadder8.top_alt.v
/* blah */
// synopsys "translate_off"
// Edit this path if necessary to reflect the actual location
// of the XilinxCoreLib library (usually $XILINX/verilog/src/
XilinxCoreLib)

`include "/XilinxCoreLib/adreVHT.v"
        // synopsys translate_on

// LIB_TAG_END ----- End LIBRARY inclusion -----

module myadder8_top (A_P, B_P, CLK_P, CE_P, CI_P, CLR_P, S_P);

input [7:0] A_P;
input [7:0] B_P;
input CLK_P;
input CE_P;
input CI_P;
input CLR_P;
output [8 : 0] S_P;

// The following block of code instantiates the core.
// 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_P),
    .B(B_P),
    .C(CLK_P),
    .CE(CE_P),
    .CI(CI_P),
    .CLR(CLR_P),
    .S(S_P));

// INST_TAG_END ----- End INSTANTIATION Template -----
endmodule

// 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); // synthesis black_box
```

```
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_P),  
        .B(B_P),  
        .C(CLK_P),  
        .CE(CE_P),  
        .CI(CI_P),  
        .CLR(CLR_P),  
        .S(S_P));
```

// synopsys translate_on

endmodule

// MOD_TAG_END ----- End MODULE Declaration -----

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.

The following example displays the Verilog parent design file: myadder8_top.v:

Code Example 2 Verilog Parent Design File: myadder8_top.v

```
//-----  
// synopsys translate_off  
// edit the next line to reflect the actual path to  
// XilinxCoreLib  
'include "/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
```

4. Create a testbench.

Write a testbench file called, testbench.v to simulate a parent design that contains the myadder8 core. Include an instantiation of the parent design and a stimulus to activate the adder. The following example displays the framework for a testbench used to simulate this design, with some sample simulation stimulus.

Code Example 3 Testbench.v File

```
'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
```

5. Analyze the behavioral simulation.

Verilog simulation netlists need to be analyzed before simulation can proceed for some vendors. For example, if you are using Model Technology's (MTI) ModelSim simulation tool to simulate your design, both the parent netlist and the testbench must be analyzed. You can analyze the simulation files by using the vlog command into a local, default, work library called work, which is created using the vlib command. Use the following commands in the project directory for MTI ModelSIM:

```
vlib work
vlog +incdir+"${XILINX}/verilog/src"test-
bench.v
vlog +incdir+"${XILINX}/verilog/src"
myadder8_top.v
vlog +incdir+"${XILINX}/verilog/src"
myadder8.v
```

You can omit vlog myadder8.v from the command line if you use the Alternate Method.

To load the testbench, the parent design, and the simulation model of the 8-bit adder core, stored in the subdirectory, Xilinx-CoreLib, invoke the simulator with the following command:

```
vsim top_level_module
```

If you are using the Cadence Verilog-XL simulation tool, you can invoke the simulator with the following command:

```
verilog +incdir+"${XILINX}/verilog/src"test-
bench.v myadder8_top.v myadder8.v
```


The `$(XILINX)` command refers to the install area. You can omit `myadder8.v` from the command line if you use the Alternate Method.

6. Synthesize the design.

Synthesize the parent design containing the core or cores. Direct the synthesizer to treat each core as a black-box. The logic for each core is specified only in its EDIF implementation netlist `component_name .EDN`, not in any Verilog file. The following table describes synthesis logic instructions for the vendor tools

Table 5-1 Synthesis Logic Instructions

Vender Tool	Instructions
Exemplar Leonardo	Do not read in a separate .V or EDIF file for the CORE Generator module. FPGA Express automatically treats the module as a black box.
Synopsys FPGA Compiler	Apply the <code>dont_touch</code> attribute to the module via the Synopsys compile script.
Synopsys FPGA Express	Do not read in a separate .V or EDIF file for the CORE Generator module. FPGA Express automatically treats the module as a black box.
Synplicity Synplify	Apply the <code>"/*</code> attribute to the component instantiation to prevent back box warning from Synplify during compilation.

The following example displays a Verilog black box:

```
// 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
```

7. Write Out the Implementation Netlist.

After the parent design has been synthesized, 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.

The 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 automatically resolves 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. Table 5-2 lists the various vendor tools and descriptions for writing out netlists.

Table 5-2 Implementation Netlist Formats

Vendor	Description
Exemplar	Write out the implementation netlist in EDIF format.
Synopsys FPGA Compiler	No special instructions. FPGA Compiler writes out an SEDIF or SXNF file for XC4000 designs, and SEDIF for Virtex designs.
Synopsys FPGA Express	No special instructions. When you direct FPGA Express to <i>Export the Netlist</i> , Express writes out an XNF file for 4K designs, and EDIF for Virtex designs.
Synplicity Synplify	Synplify writes out either XNF or EDIF netlists for both XC4000 designs. It writes out only EDIF for Virtex designs.

8. Implement the Netlist Cores.

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 Flow Engine) is run on the top level parent design during design implementation. To merge the netlists successfully, verify 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 allows you to specify explicitly the location of the directory containing the CORE Generator EDN files.

Implementation Using Cadence Verilog-XL and MTI Model Sim/VLOG

Make modifications to Cadence Verilog-XL and MTI ModelSim / VLOG third-party vendor tools using the following procedure:

1. Comment out the instantiation template in the .VEO file.
2. Copy the .VEO file to module_name.v.
3. Connect the core to the parent design by editing the module connections.
4. Replace the dummy signals in the CORE Generator module instantiation section with the actual signals in the parent design in order for the component to be connected.

The module declaration and component instantiation establishes 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 is 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.

Understanding the VHDL HDL Design Flow

This section describes the procedure for behavioral simulation, synthesis, and implementation of Verilog designs containing CORE Generator modules using the following third party vendor tools:

Third Party Vendor Tools

Third party vendor tools consist of the following:

Synthesis

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

Simulation

- MTI ModelSim / VLOG

Describing the VHDL Design Flow Procedure

This next section describes the detailed procedure for behavioral model delivery in the CORE Generator System using the VHDL design flow procedure.

1. Generating the Module

To generate the module, start the Xilinx CORE Generator by selecting

a) **Project** → **New** as shown in the following figure:

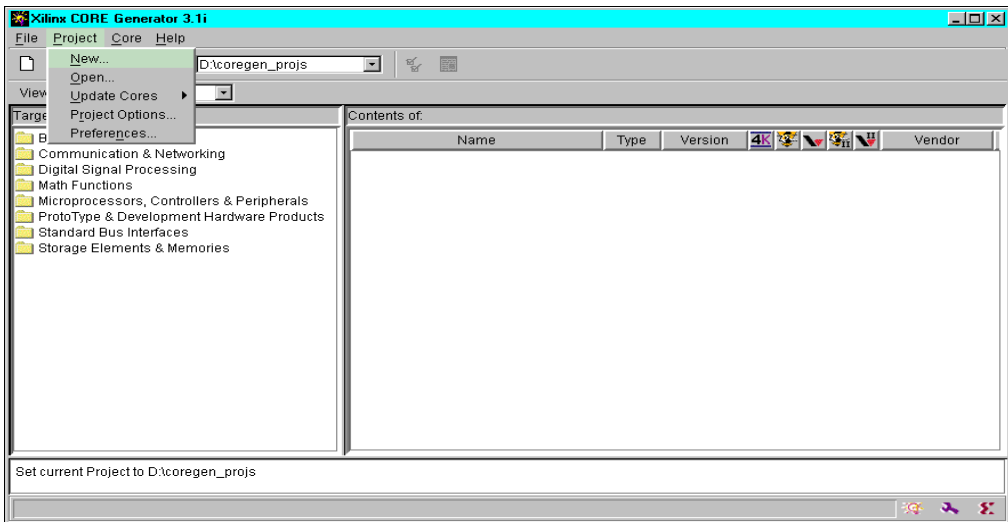


Figure 5-3 New Project Menu Screen

b) **Select Project Option** → **Design Flow** → **VHDL** → **Vendor**

This specifies the synthesis vendor software to synthesize your VHDL design. The appropriate EDIF bus delimiter format required for this flow for proper integration with the upper level parent implementation netlist is written out by the synthesizer for your design. In Figure 5-4, the Design Entry flow has been set to VHDL and the Vendor has been set to Synopsys. As a result, the bus delimiter format is automatically set to B<1>.

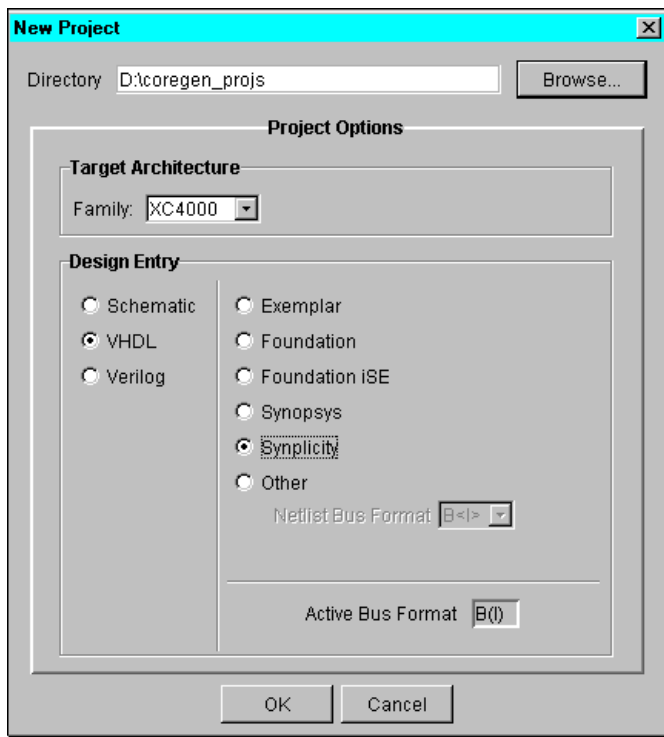


Figure 5-4 VHDL Behavioral Simulation Option

The combination of these two settings forces the bus delimiter automatically to the setting of B<I>. The following table displays the bus delimiter format settings for each vendor:

Table 5-3 Bus Delimiter Format

Vendor	Description
Exemplar Leonardo	Set Design Entry flow to VHDL, and Vendor to Exemplar. This sets the EDIF bus delimiter format to B(1).
Synopsys FPGA Compiler Synopsys FPGA Express	Set Design Entry flow to VHDL, and Vendor to Synopsys. This sets the EDIF bus delimiter format to B<1>.
Synplicity Synplify	Set Design Entry flow to VHDL, and Vendor to Synplicity. This sets the EDIF bus delimiter format to B(1).

2. Initiating VHDL Behavioral Simulation

All VHDL simulators require that the VHDL models be analyzed into the simulator's library scheme before simulation can actually proceed. The source models are located in `$XILINX/vhdl/src/vendor CoreLib`. In the specific case of Xilinx cores, the analyzed behavioral models reside in `xilinxcorelib`.

- a) Create the XilinxCoreLib library with MTI ModelSim / VHDL selected, by entering the following command:

```
cd library_directory
vlib xilinxcorelib
```

Note The name of the analyzed library must be lowercase.

- b) 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 the vendor's library directory. In your project directory, type the following:

```
vmap xilinxcorelib library_directory/xilinx
corelib
```

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

This 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 with the following command:

```
vmap xilinxcorelib full_path
_to_xilinxcorelib"
```

3. Analyzing the Behavioral Models

Analyze the vendor's VHDL models into this xilinxcorelib library in the order specified in the vhd_analyze_order file. The following excerpt is an example of the vhd_analyze_order:

```
#VHDL Simulation file list. Files are listed in
the order they should be
#analyzed in. If file F1.vhd is dependent on file
F2, then file F2 will be
#listed before F1.
```

```
#Note that all file names have been written in
lower case.
```

```
ul_tuils.vdh
mulVHT.vhd
mulVHT_comp.vhd
acc2svHT.vhd
acc2svHT_comp.vhd
```

To analyze the behavioral models in the xilinxcorelib library with MTI ModelSim / VHDL, type the following:

```
vcom -work xilinxcorelib
<path_to_Xilinx_install_dir>/vhdl/src/Xilinx-
CoreLib/ul_utils.vhd
vcom -work xilinxcorelib
<path_to_Xilinx_install_dir>/vhdl/src/Xilinx-
CoreLib/mulVHT.vhd
```

```
vcom -work xilinxcorelib
<path_to_Xilinx_install_dir>/vhdl/src/Xilinx-
CoreLib/mulVHT_comp.vhd
```

```
vcom -work xilinxcorelib
<path_to_Xilinx_install_dir>/vhdl/src/Xilinx-
```

```
Corelib/acc2svht.vhd
```

```
.....
```

```
etc.
```

Note It is critical that you compile the models in the order specified above; for example, primitive models before macro level models. Compiling the models in the wrong order leads to errors in compilation.

4. Instantiating the Module

The following procedure, for instantiating a module, is the same for all simulators.

a) Select **Design Entry** → **VHDL**

Both a VHDL template file `component_name.VHO`, and an implementation netlist `component_name.EDN` are generated whenever a CORE is generated when the VHDL option is selected in the Project Options dialog box. The `.VHO` template file includes the following items:

- ◆ Component declaration
 - ◆ Component instantiation
 - ◆ Configuration section
- b) Connect the core to the parent design by editing the instantiation block.
- c) Modify the port connections in the instantiation template to reflect the actual connections to the parent design. For more details, see the example.

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 is 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. The corresponding pins on the core are connected to the actual signal names.

The library declaration and the configuration section 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.

The `—synopsys translate_off` and the `—synopsys translate_on` compiler directives are supported by Synopsys FPGA Compiler, Foundation Express, FPGA Express, Exemplar and Synplicity synthesizers.

The specific purpose of the configuration section is to establish a link between the parent design and the core's simulation model. A separate, `module_name.VHD.VHD` file for the CORE Generator module is not needed in the project directory.

The configuration section is used only for behavioral simulation, and has no impact on the synthesis or the implementation processes. The configuration statement needs to have the following items to support behavioral simulation of the core:

- ◆ Component declaration for the core
- ◆ Library declaration for `xilinxcorelib` or the appropriate `VendorCoreLib` library
- ◆ VHDL Configuration statement

All of these components need to be added to the parent design. The corresponding configuration section for the core from the template file must be embedded in that configuration statement.

A parent design may contain multiple cores. For each core that is instantiated in the parent design, a corresponding unique configuration section must be inserted into the parent design's configuration declaration.

This next 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`.

Code Example 4 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>;
--
```

The next section consists of the parent design, myadder8_top.vhd. The component declaration, the instantiation (with dummy signal names replaced with actual signal names), and the configuration section were cut and pasted from myadder8.vho.

Code Example 5 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;
library xilinxcorelib;

```

```

-----
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.adrevVHT(behavioral)

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

5. Creating the Testbench

Write a testbench file called, `testbench.vhd` to simulate a parent design that contains the `myadder8` core and 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. The following example displays a part of the testbench file used to simulate this design. This example has one exception; that the section containing simulation stimulus is omitted.

Code Example 6 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 COMPONENT;

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_vector (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;
```

6. Performing Behavioral Simulation

Before Model Technology's simulation tools can be used to simulate the design, the parent design and the testbench need to be analyzed. These design files are analyzed with the `vcom` command, into a local, default, work library, created using the `vlib` command.

- a) Analyze the parent design and testbench file. Select the MTI ModelSim, and go to the *project_directory* and type the following:

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

- b) Invoke the simulator.

The simulator may now be invoked by typing in the following command:

```
vsim cfg_testbench
```

The `cfg_testbench` needs to correspond to the name of the VHDL configuration declared in the testbench. This loads the testbench, the parent design, and the simulation model of the myadder8 core, stored in the location referenced by `xilinx-corelib`.

7. Synthesizing the Design

You need to synthesize the parent design containing the core or cores. Direct the synthesizer to treat each core as a black-box. The logic for each core is specified only in its EDIF implementation netlist component_name.EDN, not in any VHDL file. See the following table for synthesis logic descriptions.

Table 5-4 Synthesis Logic Descriptions

Vendor Tool	Special Instructions
Exemplar Leonardo (v1998.2)	Do not read in a separate .V or EDIF file for the CORE Generator module. FPGA Express automatically treats the module as a black box.
Synopsys FPGA Compiler	Apply the dont_touch attribute to the module via the Synopsys compile script. Syntax: set_dont_touch<cell_instance_name>
Synopsys FPGA Express	Do not read in a separate .VHD or EDIF file for the CORE Generator module. FPGA Express automatically treats the module as a black box.
Synplicity Synplify	Attach a black_box attribute to component declaration for the CORE Generator module. This attribute is optional but prevents Synplicity from issuing warnings about black box modules.

Code Example 7 VHDL Black Box

```
-- 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;
```

8. Writing out the Implementation Netlist

After the parent design has been synthesized, write out its implementation netlist using the synthesis tool. This formerly created pin mismatch problems with the upper level EDIF written out by some synthesis tools. Starting with the 1.5 release of the Xilinx Implementation tools, EDIF2NGD automatically resolves 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.

Table 5-5 Implementation Netlist Formats

Vendor	Description
Exemplar Leonardo v1998.2 or later	Writes out either XNF or EDIF netlists for both XC4000 and Virtex designs. EDIF format is preferred. No other special instructions.
Synopsys FPGA Compiler	No special instructions. FPGA Compiler writes out an XNF file for 4K designs, and EDIF for Virtex designs

Table 5-5 Implementation Netlist Formats

Vendor	Description
Synopsys FPGA Express	No special instructions. FPGA Express writes out an XNF file for 4K designs, and EDIF for Virtex designs. EDIF format is preferred. No other special instructions.
Synplicity Synplify v5.1.2 and later	Synplify writes out either XNF or EDIF netlists for both XC4000 designs and Virtex designs. It writes out only EDIF for Virtex designs.

9. Implementing the VHDL 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. To merge the netlists successfully, verify 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, specifying the location of the directory containing the CORE Generator EDN files.

Appendix A

Troubleshooting the Core Generator System

This section contains solutions and resources for using the CORE Generator System.

Finding Solutions

Check the `coregen.log` file and `module_name.xco` file for diagnostic information.

- (Windows) The `coregen.log` file is located in `$XILINX/coregen/tmp`.
- (UNIX Workstations) This file is written to the current project directory.

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 the CORE Generator System. A symptom of a corrupted `coregen.prj` occurs when there are missing modules during startup causing an error message on your UNIX workstation.

LD_LIBRARY errors. 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 causes the CORE Generator System to display a detailed report of all data files being loaded and miscellaneous operations. The `-d` option causes the CORE Generator System to report specific debug-related information.

Additional Resources

For general information on Cores, use the following website:

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

Use our web-based search engine to search the Xilinx Answers Database. Please use the following Website:

<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.

To ensure that you have the latest Xilinx Software Service Packs, use the following path:

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

For the latest news on the CORE Generator System, including announcements about new IP modules and technical tips, look at the following Website:

<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.

Obtaining Customer Support

You can obtain customer support by calling Xilinx support at

1-800-255-7778 or 1-408-559-7778

or by opening a Web case at

<http://www.xilinx.com/techsup/tappinfo.htm>