CORE Generator Guide

XILINX®

About This Guide

This manual describes the Xilinx™ CORE Generator™, a design tool that delivers parameterizable cores optimized for Xilinx FPGAs.

Before using this manual, you should be familiar with the operations that are common to all Xilinx software tools: how to bring up the system, select a tool for use, specify operations, and manage design data.

Guide Contents

This book contains the following chapters.

Additional Resources

For additional information, go to http://support.xilinx.com. The following table lists some of the resources you can access from this website. 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
Answer Browser	Database of Xilinx solution records http://support.xilinx.com/xlnx/xil_ans_browser.jsp

Resource	Description/URL
Application Notes	Descriptions of device-specific design techniques and approaches http://support.xilinx.com/apps/appsweb.htm
Data Book	Pages from *The Programmable Logic Data Book*, which contains device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging http://support.xilinx.com/partinfo/databook.htm
Problem Solvers	Interactive tools that allow you to troubleshoot your design issues http://support.xilinx.com/support/troubleshoot/psolvers.htm
Tech Tips	Latest news, design tips, and patch information for the Xilinx design environment http://www.support.xilinx.com/xlnx/xil_tt_home.jsp

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
`Courier font`	Messages, prompts, and program files that the system displays	`speed grade: - 100`
`Courier bold`	Literal commands that you enter in a syntactical statement	**`ngdbuild`** *`design_name`*
Helvetica bold	Commands that you select from a menu	**File → Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *`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.

Convention	Meaning or Use	Example
Square brackets [ ]	An optional entry or parameter. However, in bus specifications, such as **bus[7:0]**, they are required.	**ngdbuild** [*option_name*] *design_name*
Braces { }	A list of items from which you must choose one or more	**lowpwr ={on|off}**
Vertical bar |	Separates items in a list of choices	**lowpwr ={on|off}**
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Repetitive material that has been omitted	**allow block** *block_name loc1 loc2 ... locn;*

Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current file or in another file in the current document	See the section "Additional Resources" for details. Refer to "Title Formats" in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the *Virtex-II Handbook*.
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

Table of Contents

Chapter 4: Batch Mode and Polling Mode

Chapter 5: Schematic and HDL Design Flows

Chapter 1

Introduction

This chapter provides an overview of the CORE Generator™ System. The chapter contains the following sections:

Overview

The CORE Generator System is a design tool that delivers parameterized cores optimized for Xilinx® FPGAs. It 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 such as filters, transforms, FIFOs, and memories.

New Features

The major new and improved features for this release include the following.

- More CORE Generator operations are integrated into Project Navigator.

  The following CORE Generator operations can now be performed from within Project Navigator, without opening the CORE Generator GUI:

  - Adding a customized core to a Project Navigator project
  - Recustomizing a core
  - Regenerating a core
  - Regenerating all of the cores in a project
  - Viewing the HDL functional model for a core
  - Viewing the CORE Generator log

  You can also open the CORE Generator window from within ISE to manage the cores in a project.

  All of these operations are described in the *ISE Guide*, the ISE online help system. The core related help topics are grouped under **FPGA Design → Using Intellectual Property (Cores)** in the help Table of Contents.

- AllianceCORE™ cores are no longer listed in the core catalog. Refer to the IP Center page on xilinx.com (http://www.xilinx.com/ipcenter) for the most up-to-date information on third party AllianceCOREs.

- The CORE Generator™ main menu structure has been enhanced to present menu selections in a manner which is more logically organized and consistent with other Xilinx® applications.
- New global preferences.
  - A global preference lets you close core customization GUIs automatically after a core has been generated.
  - A global preference controls whether cores which are scheduled to be obsoleted are displayed in the core catalog or not.
- Memory Editor improvements.
  - Support for reading and writing CSV (Comma delimited) format data files. These file are typically exported from or imported into Microsoft™ Excel spreadsheets.
  - The Memory Editor now displays the ASCII equivalent of the data value in each memory location. The ASCII equivalents are displayed in the rightmost column of the Memory Contents Panel.
  - If your memory data has a data width of 8, you can now enter an ASCII character in the ASCII column of the Memory Contents Panel and the equivalent data value will be placed in the corresponding memory address.

Xilinx Smart-IP Technology

The CORE Generator™ System creates customized cores which deliver high levels of performance and area efficiency. This is accomplished by taking advantage of Xilinx's core-friendly FPGA architectures and Xilinx Smart-IP™ technology.

Xilinx Smart-IP technology provides FPGA architectural advantages such as look-up tables (LUTs), distributed and block RAM, embedded multipliers, and segmented routing. This technology also enables relative location constraints, and expert logic mapping and floorplanning to optimize performance of a given core instance in a given Xilinx FPGA architecture.

Smart-IP technology benefits designers by providing the following features:

- Physical layout optimized for high performance
- Predictable performance and resource utilization
- Reduced power requirements achieved through compact design and interconnect minimization
- Performance independent of target device size
- Ability to use multiple instances of the same core on the same device without deterioration in performance
- Reduced compile time compared to competing architectures
- Ability to make design size and performance trade-offs
- Design flexibility

Design Flow

For each core it generates, the CORE Generator System produces an Electronic Data Interchange Format (EDIF) netlist (EDN file), a Verilog™ template (VEO) file with a Verilog (V) wrapper file, and/or a VHDL template (VHO) file with a VHDL (VHD) wrapper file. It may also create one or more NGC and NDF files. NGC files are produced for certain cores only.

The Electronic Data Netlist (EDN) and NGC files contain the information required to implement the module in a Xilinx® FPGA. Since NGC files are in binary format, ASCII NDF files may also be produced to communicate resource and timing information for NGC files to 3rd party synthesis tools. The ASY and XSF symbol information files allow you to integrate the CORE Generator™ module into a schematic design for ISE (using ECS) or for third party schematic capture tools. VEO and VHO template files contain code that can be used as a model for instantiating a CORE Generator module in a Verilog™ or VHDL design. Finally, V and VHD wrapper files are provided to support functional simulation. These files contain simulation model customization data that is passed to a parameterized simulation model for the core. In the case of Verilog designs, the V wrapper file also provides the port information required to integrate the core into a Verilog design for synthesis.

Note: The V and VHD wrapper files generated by CORE Generator cores are provided mainly to support simulation and are not synthesizable.

Figure 1-1 shows the complete CORE Generator design flow. The grayed areas in the figure 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. NGC and NDF files may also be produced for some cores. The right-side gray area shows the XilinxCoreLib and <*Vendor*>CoreLib 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.

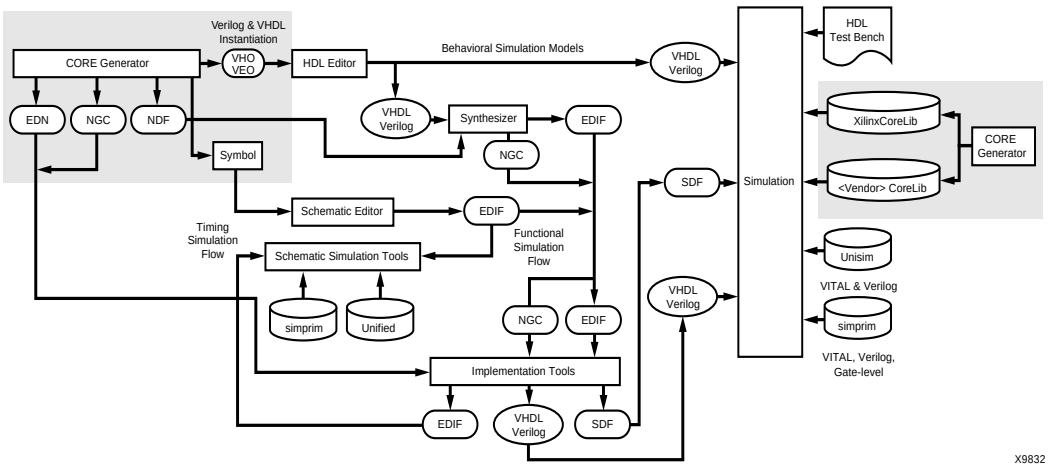


Figure 1-1: **CORE Generator Design Flow**

CORE Version Management

The CORE Generator is capable of handling multiple versions of any core or core customizer. The ability to generate new versions of a core while continuing to maintain existing versions in a design allows you to add newer cores with new functionality without disrupting the rest of the design. IP providers can also make fixes in an existing core and publish the fixes in a new version without requiring designers to use the newer version.

Each project maintains a list of cores visible to the project, each with its own version number. The cores available to the current project are displayed in the right hand panel of the main CORE Generator window and are specific to that project. This listing can be customized by the designer. When a new project is created, the latest versions of all installed cores are made visible to that project.

By default, only one version of each core is visible at any one time within a project, however multiple versions of the same core can also be made available to a project by modifying the Cores Catalog display. All cores residing in the repository are also available through batch mode.

Obsolete and Removed Cores

Some IP cores and versions of cores previously marked “To Be Obsoleted” have been removed in the 6.1i release of the CORE Generator™. This includes all XC4000 and Spartan™ cores and selected older versions of other IP.

If you need to generate or recustomize one of the removed cores, you must use the 4.2i software. A complete list of removed cores and versions of cores is available at http://www.xilinx.com/ipcenter/coregen/removed_cores/index.htm.

Chapter 2

Getting Started

This chapter describes how to start and exit the CORE Generator™. It also explains the basic elements and operations of the CORE Generator interface.

The chapter contains the following sections:

System Requirements and Installation Information

The CORE Generator is installed as part of the ISE 6.1i software from the ISE 6.1i main Xilinx® software release CDs.

The required environment variable settings are:

- XILINX variable: Set this to your Xilinx installation directory.
- PATH variable: Add $XILINX/bin/<*platform*> or %XILINX%/bin/<*platform*> (depending on platform) to your PATH variable.

Adobe Acrobat™ v 4.05 or later is needed to launch and view the core data sheets.

Some cores may require more memory than the amount specified for a particular target FPGA device in the *ISE Release Notes and Installation Guide*. See the data sheets for individual cores to find their memory requirements.

For Mentor Graphics™ eProduct™ (formerly Innoveda™) users, you can invoke the CORE Generator System interface (ePDCore) from within eProduct. This requires that both eProduct and the Xilinx implementation Tools be set up on your system. Please refer to Answer Record #11683 at the Answers Search web page for details on this interface.

See the *ISE Tutorial* or *ISE Release Notes and Installation Guide* for additional information on system requirements and installation instructions for the Xilinx software.

Starting the CORE Generator

The CORE Generator™ runs on PCs and workstations. You can start the CORE Generator from the installed ISE 6.1i software.

Starting the CORE Generator From the Windows Environment

To start the CORE Generator from your Windows® environment, select **Start** → **Programs** → **Xilinx ISE 6** → **Accessories** → **CORE Generator System**. You can also start the CORE Generator from within Xilinx® ISE (see "Starting the CORE Generator from Xilinx ISE") or from the EDA environments of Windows-based EDA tools. For example, in Innoveda™ eProduct™ v2.x DxDesigner™, you can create a new schematic, then click the Xilinx menu in Viewdraw®.

On a PC, you can also start the CORE Generator from a command prompt. To start the CORE Generator from a command prompt, select **Start** → **Run** in Windows. At the command prompt, type **`coregen`**.

Starting the CORE Generator From the UNIX Workstation Environment

At a UNIX shell prompt, type **`coregen`**. This starts the CORE Generator System.

Starting the CORE Generator from Xilinx ISE

The CORE Generator can be opened from within Project Navigator in these ways:

- If you have added a core to an ISE project, you can then open the CORE Generator GUI from within the ISE Project Navigator. Project Navigator will run on a PC or a UNIX workstation.

  To open the CORE Generator GUI from within Xilinx ISE:

  a. In Project Navigator, select an IP core name in the **Sources in Project** window.

  b. Click the "+" icon next to the **Coregen** process in the **Processes for Current Source** window.

     The **Manage Cores** process shows in the processes window.

  c. Double-click on **Manage Cores**.

  The CORE Generator window displays.

- A number of CORE Generator operations can be performed within Project Navigator without opening the CORE Generator window. These operations are described in "Performing CORE Generator Operations in Xilinx ISE" in Chapter 3.

Setting Preferences

Your preferences are set through the Preference Options dialog box (see Figure 2-1), which is opened by selecting **File** → **Preferences**. Preferences are maintained on a per user basis.

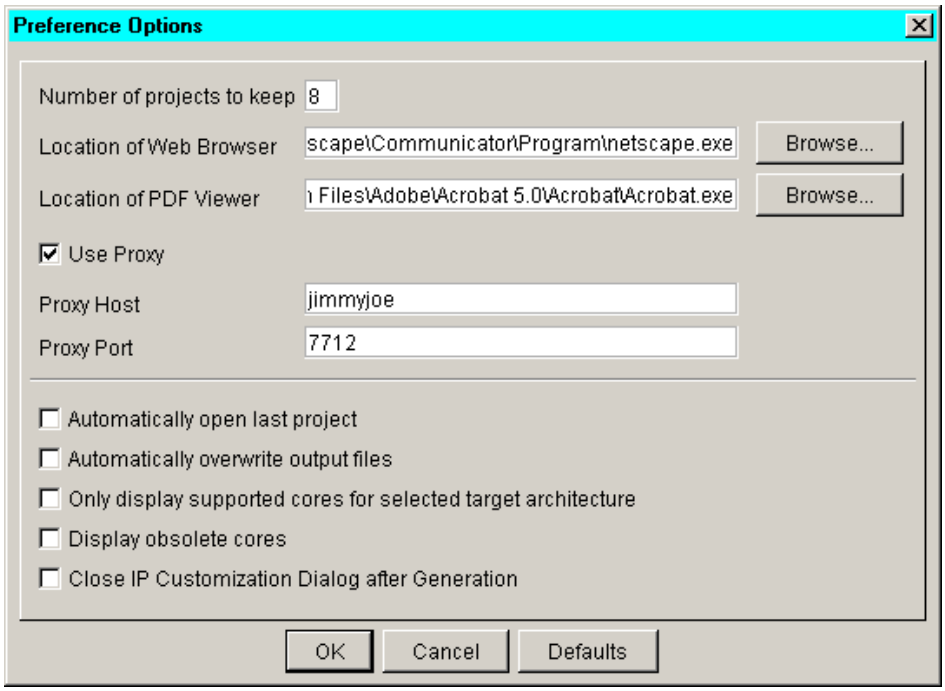


Figure 2-1: **Preference Options Dialog Box**

The location at which preferences are saved on various platforms is as follows:

- Windows® – Preferences are stored in the Windows registry.
- UNIX® workstation – Preferences are stored in your home directory in the .coregen.prf file.

For information about how to set preferences, see "Setting Preferences" in Chapter 3.

Command Line Options

The CORE Generator™ System is invoked in batch mode as follows:

```
coregen -b <command_file_name> -p <project_path>
```

Table 2-1: **Command Line Options**

Option	Definition
-b *<command_file_name>*	Invokes the CORE Generator in batch mode and the name of the command file that should be executed during the batch mode run. The *command_file_name* argument specifies the path to the command file to be executed. XCO files are commonly specified as the argument to the **-b** option, but you can specify any file containing valid CORE Generator commands.
-i *<coregen_ini_file_name>*	When the **-i** option is used, the CORE Generator looks for the specified INI file in the current working directory if no path is specified. If a different INI profile is required, then the path can be explicitly specified using either an explicit or relative path name. The *coregen_ini_file_name* is the path to the CORE Generator INI file to be loaded. If the **-i** option is not specified, the CORE Generator System looks for an INI profile in the current working directory by default.
-p *<project_path>*	Specifies the CORE Generator project directory. The *project_path* argument is the path to the desired CORE Generator Project. This path can be specified relative to the CORE Generator startup directory.
-q *<polling_dir_path>*	This is an option for third party tools that call the CORE Generator System in polling mode. Do *not* use in batch mode. The *polling_dir_path* supplied is the path to the polling directory where the polling mode communication files are written.

Table 2-1: **Command Line Options**

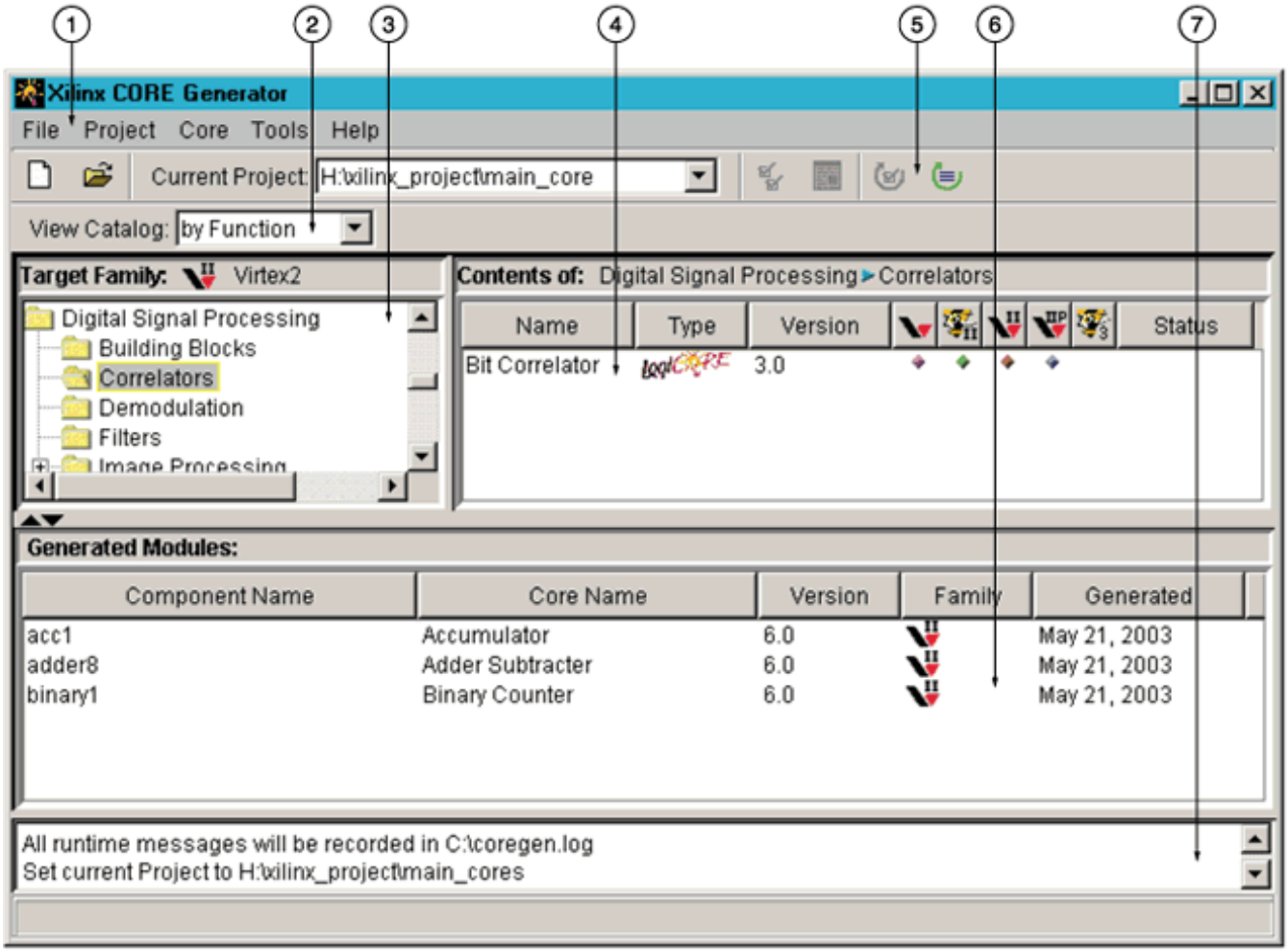
Option	Definition
-intstyle <**ise** | **xflow** | **silent**>	Determines the integration style in which the CORE Generator™ should run. The **ise** and **xflow** arguments direct the CORE Generator to run in a mode compatible with ISE or XFLOW, with limited messaging and application identification output. However, in the 6.1i release this behavior has not been implemented, so there is no difference between the default operation of CORE Generator and the operation when invoked with **-intstyle ise** or **-intstyle xflow**. The **silent** argument opens the CORE Generator GUI without displaying a splash screen.
-h	Displays the CORE Generator batch mode command line help and version information.
-d	Invokes CORE Generator in debug mode. Directs the CORE Generator to generate verbose runtime messaging.

Using the Interface

This section describes the CORE Generator™ graphical user interface and how to use it.

Main Window

This section describes the CORE Generator main window (shown following).



1. Menu Bar
2. View Catalog Toolbar
3. and 4. Cores Catalog Browser
5. Standard Toolbar
6. Generated Modules Window
7. Console Window

Menu Bar

The menu bar is located under the window's title bar (which says **Xilinx CORE Generator**).

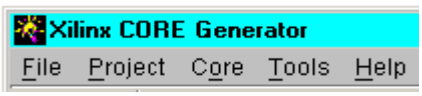


Figure 2-2: **Menu Bar Selections**

You can select menu commands with the mouse or the keyboard. With the mouse, click the left mouse button on the command. With the keyboard, press the **Alt** key and type in the letter underlined in the menu for that command.

Some menu commands include an ellipsis (...). When you select one of these commands, a dialog box appears.

Standard Toolbar

The Standard toolbar contains commands to perform these common operations on your designs:

- Creating a new project
- Opening an existing project
- Selecting the current project
- Customizing a core
- Viewing the data sheet for a core
- Recustomizing or regenerating a core

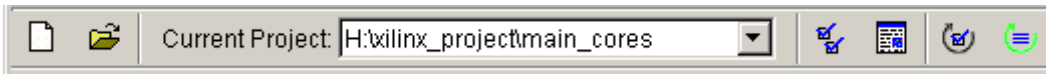


Figure 2-3: **Standard Toolbar**

View Catalog Toolbar

The View Catalog toolbar allows you to choose how the available cores are displayed in the CORE Generator™ window.

Figure 2-4: **View Catalog Toolbar**

Cores can be displayed in the following ways:

- **by Function** (default)

  Displays the cores in folders, sorted according to function.

- **Alphabetically**

  Displays the cores in alphabetical order.

- **by Vendor**

  Displays the cores by vendor (IP provider).

- **by Family**

  Displays the cores by device family.

- **by Type**

  Displays separate listings for LogiCORE™ cores and Reference Designs

Cores Catalog Browser

The Cores Catalog Browser displays the cores that can be customized and included in your CORE Generator™ project.

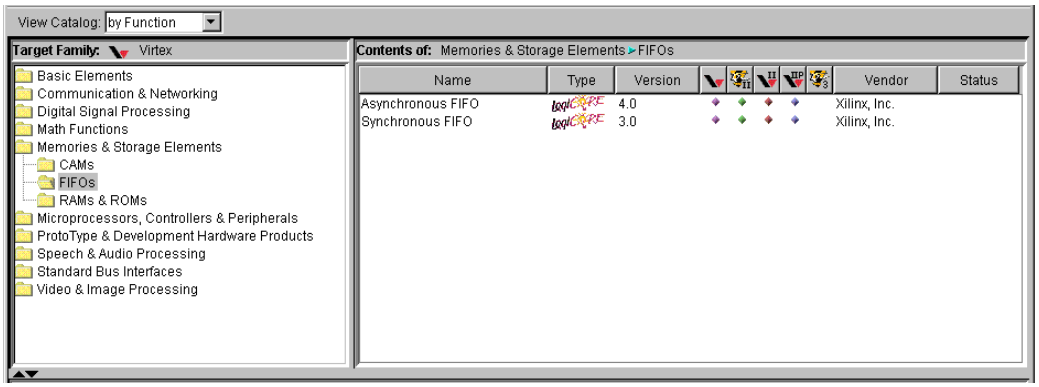


Figure 2-5: **Cores Catalog Browser**

The left hand portion of the Cores Catalog Browser, the Core Browser Navigation Panel, contains a set of folders arranged hierarchically. The folders displayed in the Core Browser Navigation Panel can be organized by core Function, by Type (LogiCORE or Reference Design), by Vendor (IP provider), by device Family, or Alphabetically (see "View Catalog Toolbar").

The right hand portion of the Cores Catalog Browser, the Core Browser Contents Panel, displays the selection of core customizers within the folder selected in the Core Browser Navigation Panel. In the Core Browser Contents Panel you can select a core to customize it, view its data sheet, or view version information for it.

For a complete description of the Cores Catalog Browser, see "Using the Cores Catalog Browser" in Chapter 3 and "Configuring the Cores Catalog Browser" in Chapter 3.

Generated Modules Window

The Generated Modules window displays the Component Name, Core Name, Version, Family, Vendor and date generated of each core generated.

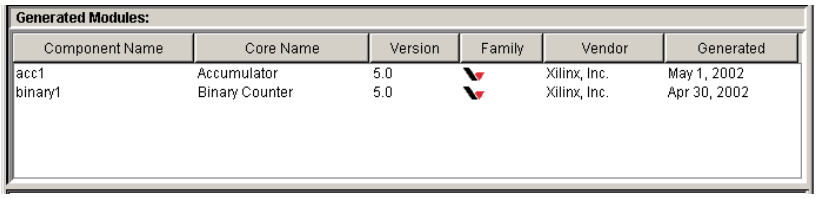


Figure 2-6: **Generated Modules Window**

Console Window

The Console Window displays commands and responses. All error messages, warnings, and command responses are written to the Console Window.

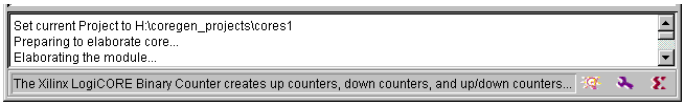


Figure 2-7: **Console Window**

Use the scroll bar located to the right of the console window to view all the commands and responses recorded during an editing session. The Console Window can be resized to show more lines of messaging at once. Move your mouse anywhere on the upper border of this window until the cursor turns into a two-headed arrow, then resize the window as desired.

Using Dialog Boxes

Many menu commands display dialog boxes in which you can enter information and set options.

Using Common Fields

The fields shown in the following table are common to most dialog boxes.

Table 2-2: **Common Dialog Box Fields**

Dialog Box Field	Function
OK	Closes the dialog box and implements the intended action according to the settings in the dialog box.
Dismiss	Dismisses the core customization dialog when you are finished customizing cores with that GUI.
Apply	Implements the intended action.
Cancel	Closes the dialog box.
Select	Displays various Target architectures.
Set	Displays True, False, Default option for the Overwrite Files category.
Reset	Changes all settings back to their default values.

Browse Buttons

Many dialog boxes contain browse buttons to allow you to navigate through your directory structure to find a particular file or to save a file to a specific location.

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

Other options available from the **Help on the Web** menu are links to these locations:

- The CORE Generator Examples web page
- Xilinx® Support and Services
- The main Xilinx corporate web page at www.xilinx.com

Accessing Core Data Sheets

To view the data sheet for a core:

1. Select a core by clicking the name of the core in the Core Browser Contents Panel.
2. Select **Core** → **Data Sheet** or click the Data Sheet button on the standard toolbar.

The Acrobat™ Reader displays the data sheet.

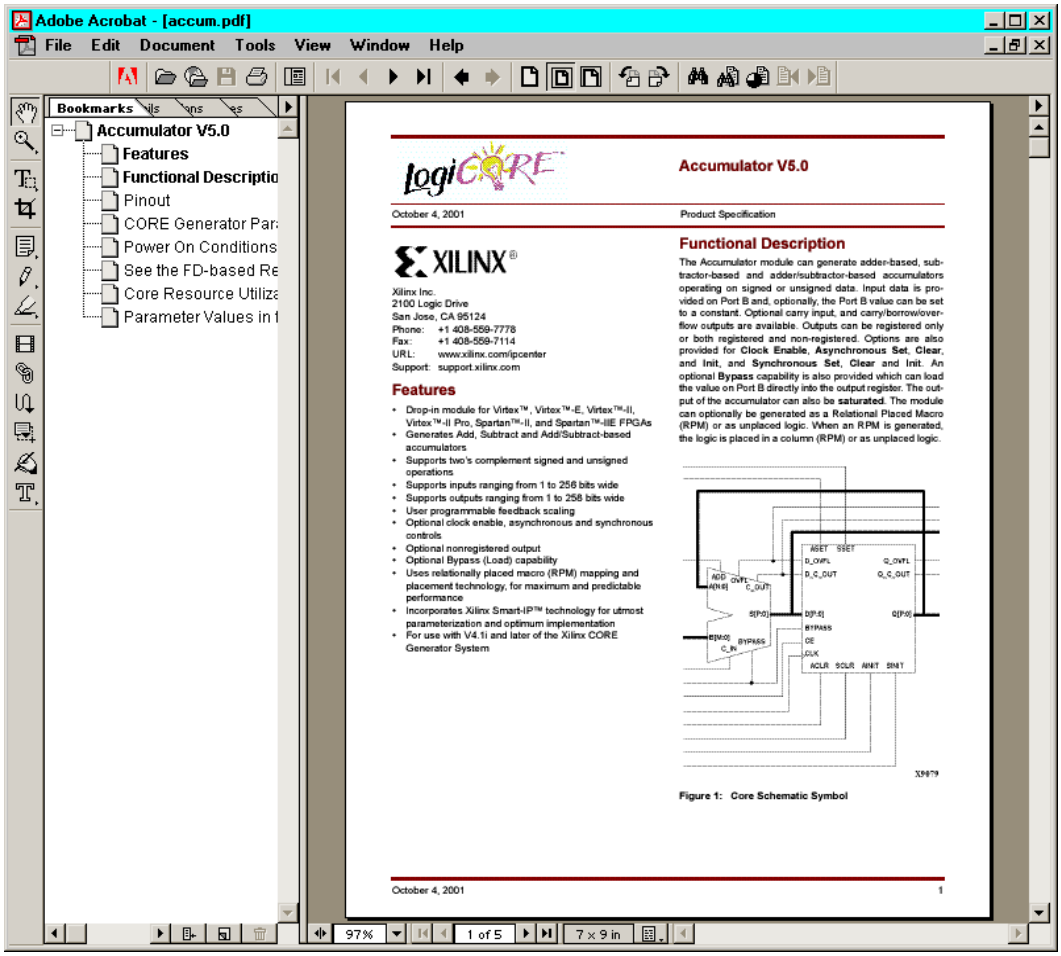


Figure 2-8: **CORE Generator Data Sheet**

You can also access a data sheet by right clicking a core in the Core Browser Contents Panel. Data sheets can be viewed for any core listed in the Core Browser Contents Panel whether they are grayed out or not.

Chapter 3

Using the CORE Generator

This chapter explains the major functions a designer performs when using the CORE Generator™. The chapter contains the following sections:

Using the Cores Catalog Browser

The Cores Catalog Browser is located in the upper panel of the CORE Generator main GUI. Cores that fall into particular functional categories are grouped into folders in the Cores Catalog Browser to assist you in locating the core appropriate to your needs. The left hand panel of the Cores Catalog Browser, the Core Browser Navigation Panel, allows you to browse through these folders. To select a folder, click once on the folder name in the Core

Browser Navigation Panel. To expand a folder, double-click the folder icon to the left of the folder name. To close a folder, double-click the open folder icon. Some folders have a + icon or a – icon to their left. The + indicates the folder has subfolders that are not displayed, and the - indicates the subfolders are displayed. You can open or close the folder with a single click on those icons.

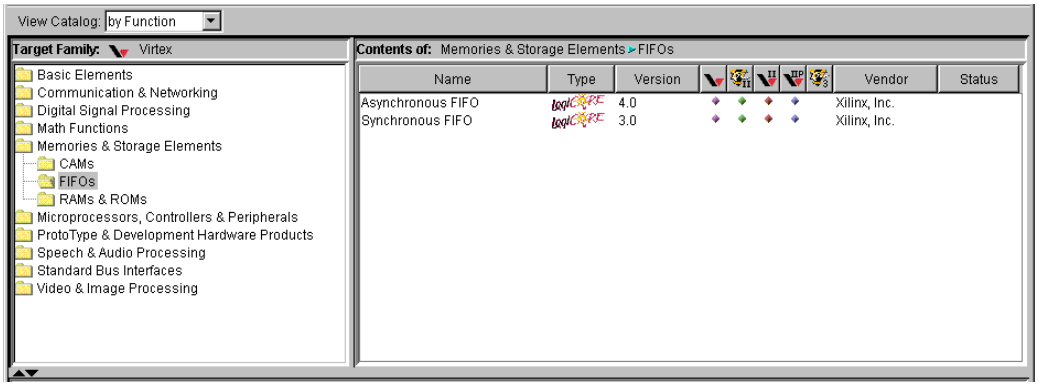


Figure 3-1: **Cores Catalog Browser**

The cores in the selected folder display in the right hand panel of the Cores Catalog Browser, the Core Browser Contents Panel. Within a folder, cores are listed alphabetically by name and also have type, version, family and vendor information displayed in columns.

Core status information is displayed in the far right column, and may include one of the following icons:

Icon	Meaning
	Core is scheduled to be obsoleted. If you don't want to view the cores which are scheduled to be obsoleted, a global preference (**Display obsolete cores**) allows you to select whether these cores are displayed or not (see "Setting Preferences").
	Core requires an additional license before it can be used.

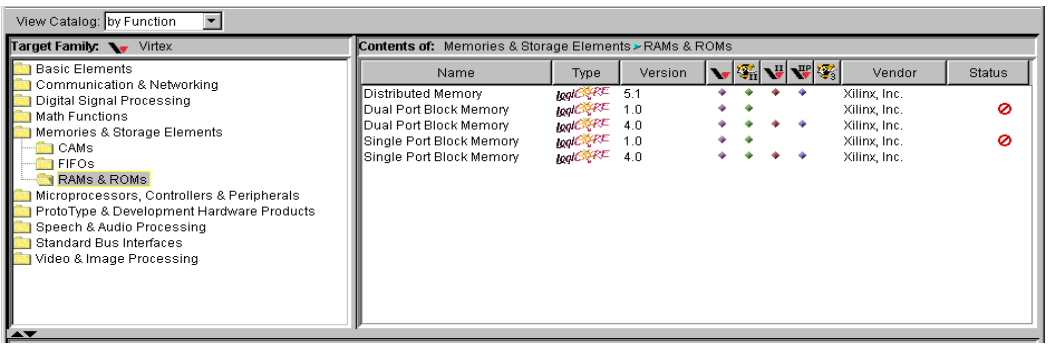


Figure 3-2: **Example of Cores Catalog with Obsoleted Cores**

Some cores in the Core Browser Contents Panel may be grayed out. This means that these cores are not available for the currently selected Xilinx® FPGA family.

Sorting the Catalog

You can sort the cores in the Cores Browser Navigation Panel in a number of different ways by setting the View Catalog setting at the top of the Cores Catalog Browser. Cores may be listed by Function, Alphabetically, by Vendor (IP provider), by Family or by Type (LogiCORE™ or Reference Design).

Adjusting Columns and Panels

The size of each column within the Cores Catalog Browser can be adjusted by moving the separators between the column headings. Panels can be resized in a similar way.

A panel will have a vertical or horizontal scroll bar (or both) to allow navigation if the information displayed in the panel is larger than the current panel size.

Using the Generated Modules Window

Cores that have been generated in a project are displayed directly under the Cores Catalog Browser in the Generated Modules window. When you double click a core in this panel, you can perform either of these functions:

- **Recustomize** – Allows you to call up a previously generated core with the original parameters used to generate it, then modify these parameters and generate a new version of the core. You can recustomize under the original project settings or under the current project settings. See "Recustomizing a Core".
- **Regenerate** – Allows you to regenerate a core to create a different set of output products. You can regenerate under the original project settings or under the current project settings. See "Regenerating a Core".

Accessing New and Updated Cores

Up to date information on the full range of Xilinx® IP Solutions is available on the Xilinx IP Center page at http://www.xilinx.com/ipcenter.

On the IP Center page you can find information on the latest general release IP updates from Xilinx (downloadable free of charge), as well as information on more complex system level IP which you can evaluate and purchase. Also available are links to Reference Design resources and third party consultants.

The IP Center web page is updated on a regular basis, so be sure to review it before starting a new design to make sure you are aware of the latest IP offerings available from Xilinx.

Figure 3-3 shows how you can access the IP Center page directly from the Help menu in the CORE Generator™ GUI.

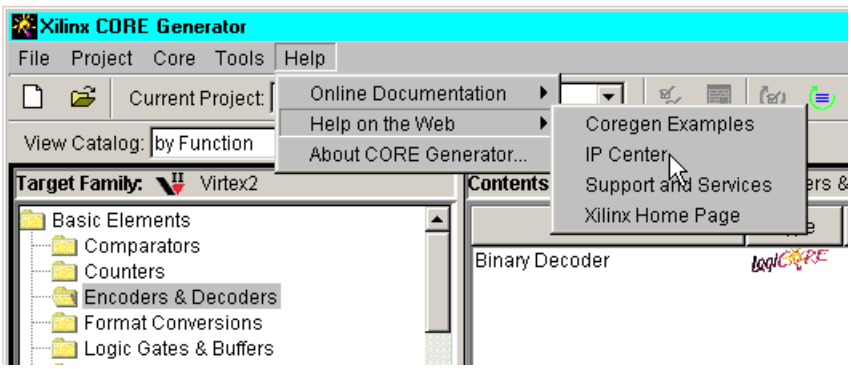


Figure 3-3: **Navigating to the Xilinx IP Center from the CORE Generator Window**

Installing New Cores

When you download new cores and new versions of existing cores from the IP Center, they are installed in the CORE Generator's built-in IP repository 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 associated with new cores 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 direct link to new Xilinx® standard release cores and their installation instructions is available on the Xilinx website at this location:

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

After new cores have been added to your CORE Generator IP repository and a project is opened, a dialog box appears, asking whether you wish to update your project's list of visible cores (see Figure 3-6). You may choose:

- **All**

  Only the latest version of all cores will be visible in the Cores Catalog display.

- **New**

  The latest version of all new cores will be visible in the Cores Catalog display.

- **Custom**

  Allows you to customize which cores are made visible in the Cores Catalog display.

- **None**

  None of the new cores will be visible in the Cores Catalog display.

Working With Licensed Cores

In the 6.1i release, CORE Generator™ LogiCORE™ cores which require an additional license may be included in the release CD, or added via subsequent IP Updates. There are two types of licenses that may be required for these types of cores:

- Full System Hardware Evaluation licenses
- Full Release IP licenses

Full System Hardware Evaluation licenses are offered on some of the higher complexity, system level cores. This license enables you to perform a Full System Hardware Evaluation of the core. A core which supports Full System Hardware Evaluation allows you to:

- Integrate the core into the rest of your design
- Process the design through map, place and route
- Generate a bitstream
- Program the design into your target Xilinx® FPGA
- Perform timing simulation and static timing analysis
- Review all documentation found in the full product offering

If you generate a bitstream and then program an FPGA using a core that has a Full System Evaluation license, the core will stop working in the programmed device after 2-8 hours, depending on the core. To get the device working again you must reload the bitstream, reprogramming the device.

Full Release IP licenses allow you to access the full functionality of the official released version of a licensed core.

In the CORE Generator, if you double click on a core in the Cores Catalog Browser that requires a license for evaluation or full functionality, a message box for this core will indicate that you can only generate functional models for the core, unless you obtain an additional license.

Licenses for evaluation and full functionality access can be requested from the product lounge for the core on Xilinx's IP Center web page at this location:

http://www.xilinx.com/ipcenter

To obtain a license you will typically need to register for the lounge and fill out an online request form. For full release core licenses you will also need to provide a serial number. Consult the respective product lounge in the IP Center for specific instructions, and follow the instructions for installing the license. Once you have done this, you should be able to use the core in either evaluation or full release mode.

Creating a New Project

This section describes how to create a new project. When a new project is created the cores displayed in the CORE Generator System's main window are the latest versions of the cores.

To create a new project in CORE Generator:

1. Select **File** → **New Project** or click the New Project toolbar button.

The New Project dialog box appears, as shown in the following figure.

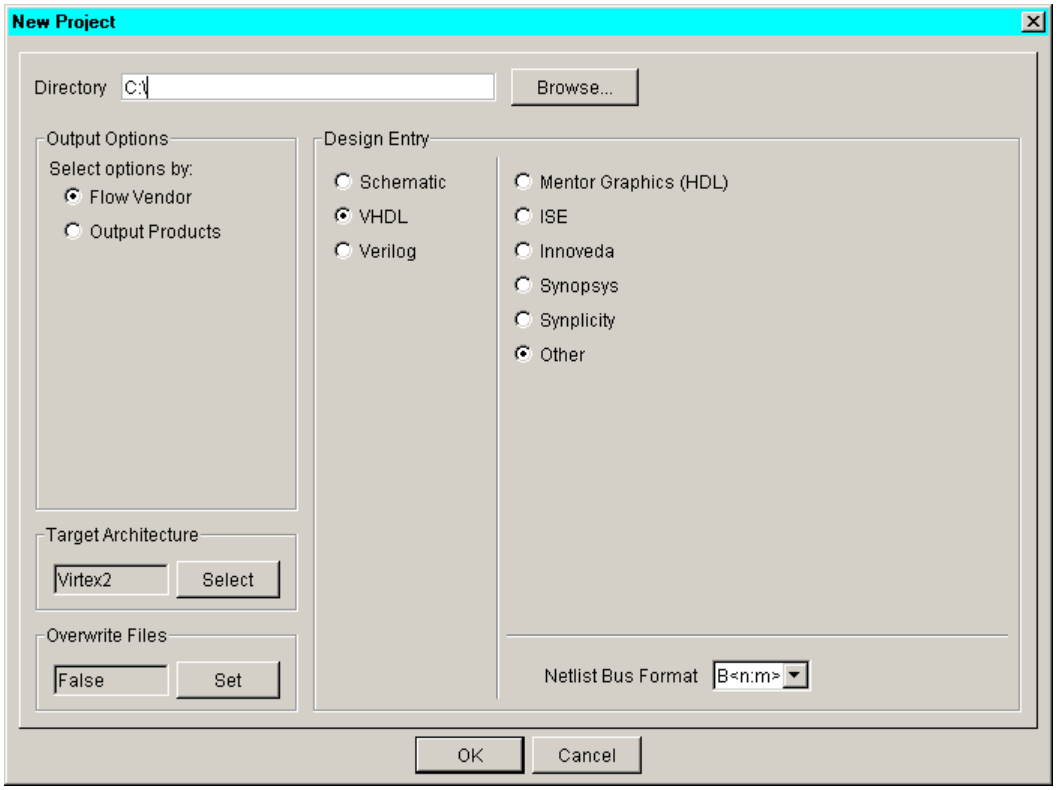


Figure 3-4: **New Project Dialog Box**

2. In the New Project dialog box, type the path to the new project directory in the **Directory** text field

   OR

   Click the **Browse** button and navigate to the project directory.

3. Select your project options as directed in "Changing Project Options".

   Note: You cannot create a new CORE Generator™ project targeted to an ISE flow. If you select **ISE** in the New Project dialog box, you will receive an error message directing you to create a new project within ISE instead of within the CORE Generator.

4. 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 directory. The coregen.prj file contains a record of all installed cores at the time the project was created and their latest versions.

5. Set the rest of your project options as directed in "Changing Project Options".

Opening an Existing Project

To open an existing project from within CORE Generator™:

1. Select **File** → **Open Project** or click the Open toolbar button.

The Open project box displays.

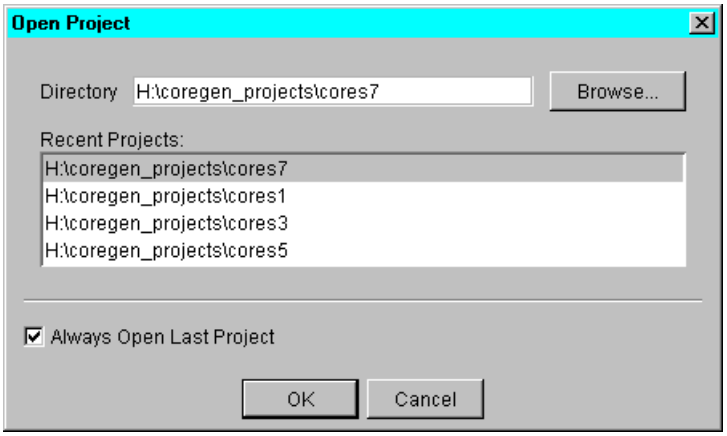


Figure 3-5: **Open Project Box**

2. In the Open Project dialog box, select a project from the Recent Projects list in the dialog box

   OR

   Click the **Browse** button and navigate to the project directory.

   You may also place a check mark in the **Always Open Last Project** check box in the following figure. If you select this box, the CORE Generator System bypasses the Open Project dialog box at startup, and opens the last open project. If you deselect the **Always Open Last Project** check box, CORE Generator prompts you for a project at startup by displaying the Open Project dialog box.

3. Click **OK**.

   If the project has been locked by another user, the CORE Generator gives you the option of Closing the project, displaying More Info about the project lock, or Removing the lock. Displaying More Info gives you more information on who has locked the project and also gives you the option of removing the lock.

   If new IP have been added to the CORE Generator repository since you last accessed the project, the following dialog box appears.

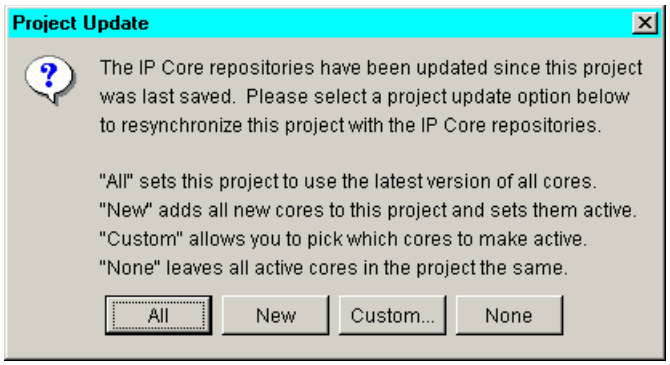


Figure 3-6: **Project Update Dialog Box**

The dialog box presents options for updating the list of visible cores in the CORES Catalog for the project. The selections in this dialog box are described in "Installing New Cores".

Changing Project Options

To change Project Options:

1. For an existing project, select **Project** → **Project Options**. This opens the Project Options dialog box for the project.

   For a new project, a similar dialog box displays. The dialog box is titled New Project, and has an additional field at the top allowing you to specify the project location.

2. Under the Output Options panel, select either the **Flow Vendor** or **Output Products** view. The Flow Vendor view allows you to specify your Design Entry flow (Schematic, VHDL, or Verilog) and Design Entry vendor, and then determines the appropriate output products for the cores you generate automatically.

   The Output Products view is for more advanced users. This view allows you to explicitly specify the output products you wish to be generated for each core you create.

3. Modify the project options in the Project Options dialog box.
   - Project options for the Flow Vendor view are described in "Output Options – Flow Vendor".
   - Project options for the Output Products view are described in "Output Options – Output Products".

4. When you have finished modifying the project options, click **OK**.

   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.

Output Options – Flow Vendor

After you have selected the Flow Vendor view, you can then set the Target Architecture, Overwrite Files option, the Design Entry flow and Design Entry vendor.

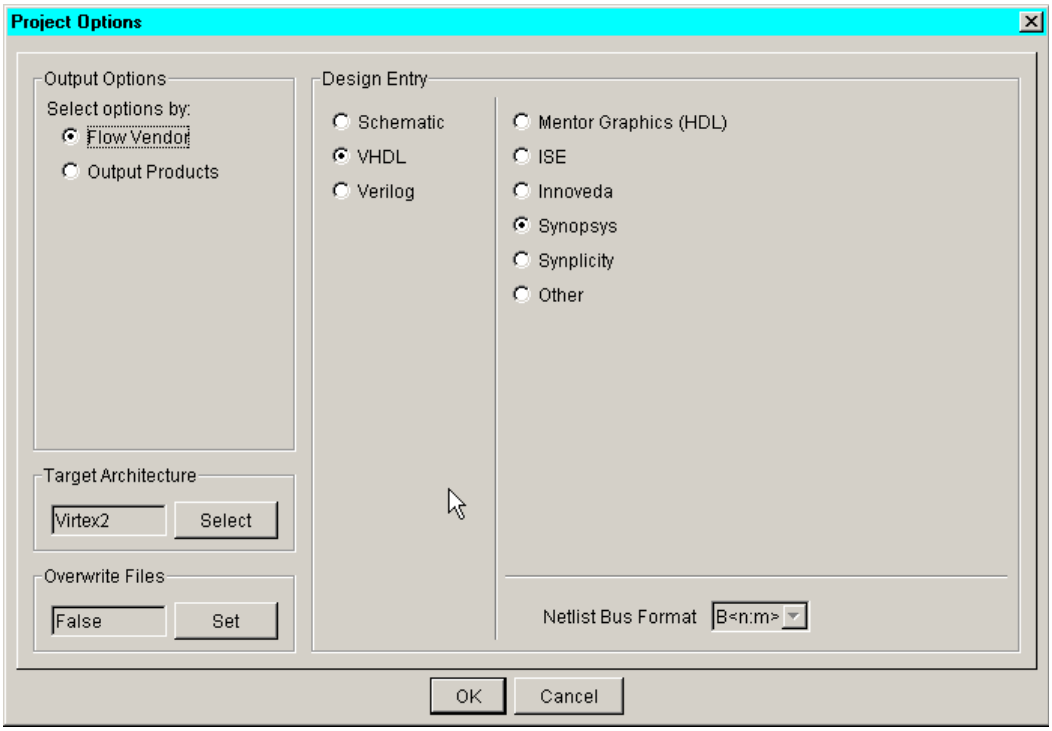


Figure 3-7: **Project Options – Flow Vendor**

Target Architecture

Select your target architecture from this list:

Table 3-1: **Target Architecture**

Design	Specification
Virtex	Virtex, VirtexE
Spartan2	Spartan2, Spartan2E
Virtex2	Virtex2
Virtex2P	Virtex2P
Spartan3	Spartan3

Overwrite Files

You can select these options for Overwrite Files:

- **True**

  Any pre-existing files associated with a previously generated core are overwritten by default with no warning when a new core of the same name is generated.

- **False**

  The CORE Generator™ prompts you for approval before overwriting existing files.

- **Default**

  The Overwrite Files behavior follows the global preference setting specified for all projects as set in Preference Options dialog box (accessed by selecting **File** → **Preferences**).

Design Entry

Specify your Electronic Design Automation (EDA) flow (Schematic, VHDL or Verilog™) and design entry vendor.

If you choose **Schematic** as the design flow, the following outputs are generated depending on your chosen vendor:

- **Cadence**

  The CORE Generator produces a Cadence™ compatible EDN file.

- **ISE**

  The CORE Generator produces an ASCII symbol (ASY) file, an ECS symbol (SYM) file (for the Xilinx ECS schematic editor), and the underlying EDN netlist.

- **Innoveda**

  When run standalone, the CORE Generator does not generate any schematic output if you select **Innoveda**, however it does create an eProduct™ compatible EDN file. When the CORE Generator is invoked through the Xilinx / Innoveda™ interface (ePDCore), ePDCore automatically creates a .1 symbol file, which can be instantiated in a Viewdraw schematic sheet. For more details on the Innoveda Xilinx™ flow, see Answer Record #11683 at the Xilinx Answers Search web page.

- **Mentor Graphics (Schematic)**

  The CORE Generator produces an XSF file to support symbol generation within the Mentor™ Design Architect™ environment, and a Mentor compatible EDN netlist.

- **Other**

  The CORE Generator produces an EDN netlist, with the Netlist Bus Format you specify.

When you choose **VHDL** as the design flow, a VHO instantiation template file and a VHD wrapper file are created. The VHO template file contains commented HDL example code that can be used to instantiate a CORE Generator module in an HDL design. The VHD wrapper file is used to support functional simulation of the core. Similarly if you choose **Verilog** as the design flow, a VEO instantiation template file and V wrapper file are created.

Note: If you select any vendor except **Other**, the CORE Generator automatically sets the Netlist Bus Format setting in the output EDIF file to the correct value for that vendor. If you set the Design Entry vendor to **Other,** you will also need to specify the Netlist Bus Format.

Netlist Bus Format

Sets the format in which bus signals are written in the output EDIF file. If you select **Other** as your design entry vendor, you will have to set a Netlist Bus Format.

Bus format can be specified (and will be written into the EDIF file) in either of these ways:

- As individual bus bits. For these options, B represents the name of the bus and I represents the bus index. Options are B<I>, B(I), B[I], and BI.
- As a single array. For these options, B represents the name of the bus and n:m represent the range of the bus index. Options are B<n:m>, B(n:m), and B[n:m].

Design Flows Supported by the CORE Generator

The following table lists the design flows supported by the CORE Generator on a vendor by vendor basis.

Table 3-2: **Design Entry Vendors and Design Entry Flow Options**

	Schematic	VHDL	Verilog
Cadence™	X		X
Innoveda™	X	X	X
ISE	X	X	X
Mentor Graphics™ (HDL)		X	X
Mentor Graphics (Schematic)	X		
Synopsys™		X	X
Synplicity™		X	X
Other	X	X	X

Output Options – Output Products

The Output Products option in the Project Options dialog is geared toward the more advanced user. In addition to allowing you to select your target architecture and File Overwrite options, it allows you to choose the exact output products you need more precisely. It also supports some additional features such as formal verification outputs and three special elaboration options.

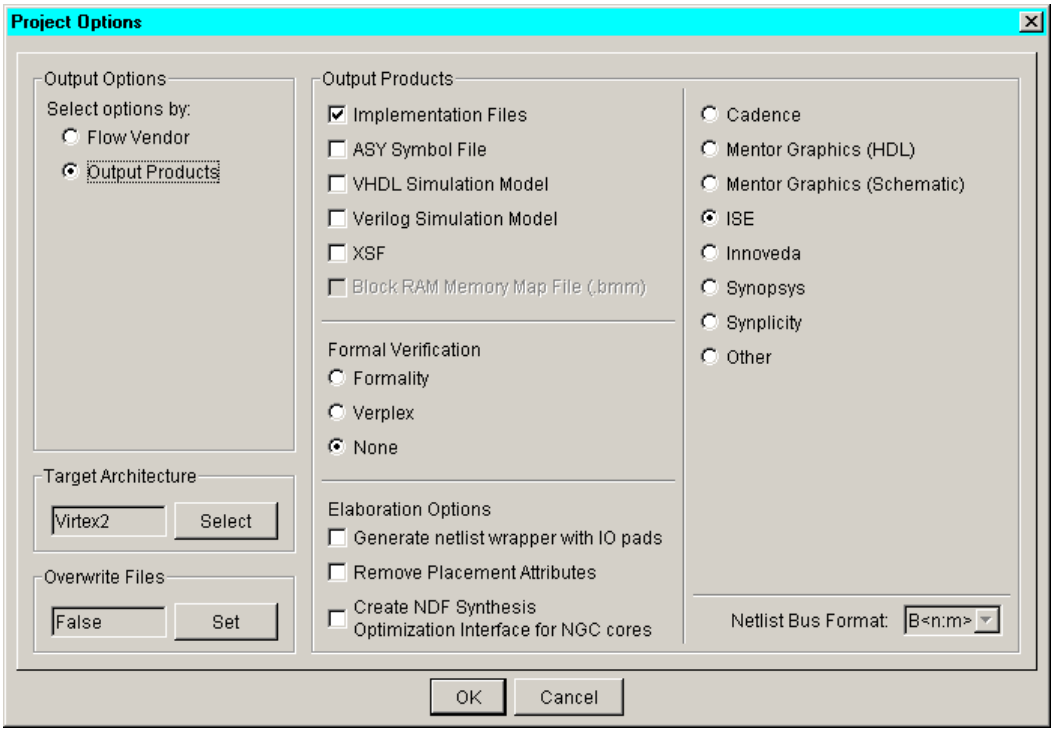


Figure 3-8: **Project Options – Output Products**

Target Architecture

Select your target architecture from this list:

Table 3-3: **Target Architecture**

Design	Specification
Virtex	Virtex, VirtexE
Spartan2	Spartan2, Spartan2E
Virtex2	Virtex2
Virtex2P	Virtex2P
Spartan3	Spartan3

Overwrite Files

You can select these options for Overwrite Files:

- **True**

  Any pre-existing files associated with a previously generated core are overwritten by default with no warning when a new core of the same name is generated.

- **False**

  The CORE Generator prompts you for approval before overwriting existing files.

- **Default**

  The Overwrite Files behavior follows the global preference setting specified for all projects as set in Preference Options dialog box (accessed by selecting **File** → **Preferences**).

Output Products

When you select the Output Products view of the Project Options dialog, you can choose from the following selections from the Output Products panel:

- **ASY Symbol File**

  An ASCII symbol information file used by the ISE tools and some third party interface tools to create a symbol representing the core.

- **VHDL Simulation Model**

  Generates files for simulating a generated core in a VHDL simulation environment.

- **Verilog Simulation Model**

  Generates files for simulating a generated core in a Verilog™ simulation environment.

- **XSF**

  Symbol information file for Mentor.

- **Block RAM Memory Map File (.bmm)**

  This option, which applies to Block Memory cores, is currently disabled and cannot produce any BMM output files. This feature may be enabled in a future 6.1i IP update.

EDIF implementation files are always generated by default. For some cores, NGC files are also generated.

The right hand panel of the Output Products dialog view lists these supported vendors:

- **Cadence**
- **Mentor Graphics (HDL)**
- **Mentor Graphics (Schematic)**
- **ISE**
- **Innoveda**
- **Synopsys**
- **Synplicity**
- **Other**

Note: If you select any vendor except **Other**, the CORE Generator automatically sets the Netlist Bus Format setting in the output EDIF file to the correct value for that vendor. If you set the Design Entry vendor to **Other**, you will also need to specify the Netlist Bus Format (see "Netlist Bus Format").

Formal Verification

There are three options to choose.

- **Formality**

  Generates a Verilog™ model called <*module_name*>_for.v to support formal verification using the Formality™ tool from Synopsys™.

- **Verplex**

  Generates a Verilog model called <*module_name*>_for.v to support formal verification using the Tuxedo-LEC™ Logic Equivalence Checker tool from Verplex.

- **None**

  No files are generated to support formal verification.

Elaboration Options

There are three options to choose.

- **Generate netlist wrapper with IO pads**

  This option adds input pads and output pads to a core when the CORE Generator™ generates the core and writes out its EDIF implementation netlist. Use this file if you want to process the core standalone all the way through place and route to get precise timing and resource utilization information. You can do this without having to interface to any design entry tool.

  If **Generate netlist wrapper with IO pads** is selected, the Core Generator creates an additional file, a "padded" EDIF wrapper file. The padded EDIF file contains a declaration of the core as a black box, with the ports connected to appropriate Input/Ouput blocks.

  For a core named *corename* the EDIF describing the core is generated as usual in a file named *corename*.edn. The additional file generated when **Generate netlist wrapper with IO pads** is selected is named *corename*_padded.edn. The module defined in the file is named *corename*_padded.

  The *corename*_padded definition includes a black box instantiation of the core, according to the definition in *corename*.edn. Each strand of each port on the core is connected to an appropriate Input/Output Block (IOB). Each IOB is connected to the corresponding port strand on the *corename*_padded module.

  IOBs are added according to these rules:

  - The IOB type connected to a port is shown in the following table:

Table 3-4: **Port Connections for Generate netlist wrapper with IO pads Option**

Port Type	IOB(s)
Output	OBUF
Input Clock (Virtex™ derivatives)	IBUFG connected to a BUFG
Input Clock (all other families)	BUFG
All other Input Ports	IBUF
Bidirectional	IOBUF

- A clock port is assumed if a port is named clk or g, or starts or ends with clk or _ck.
- IPADs and OPADs are *not* added.

The following figure gives an example of the padded file generated for a Virtex™ core.

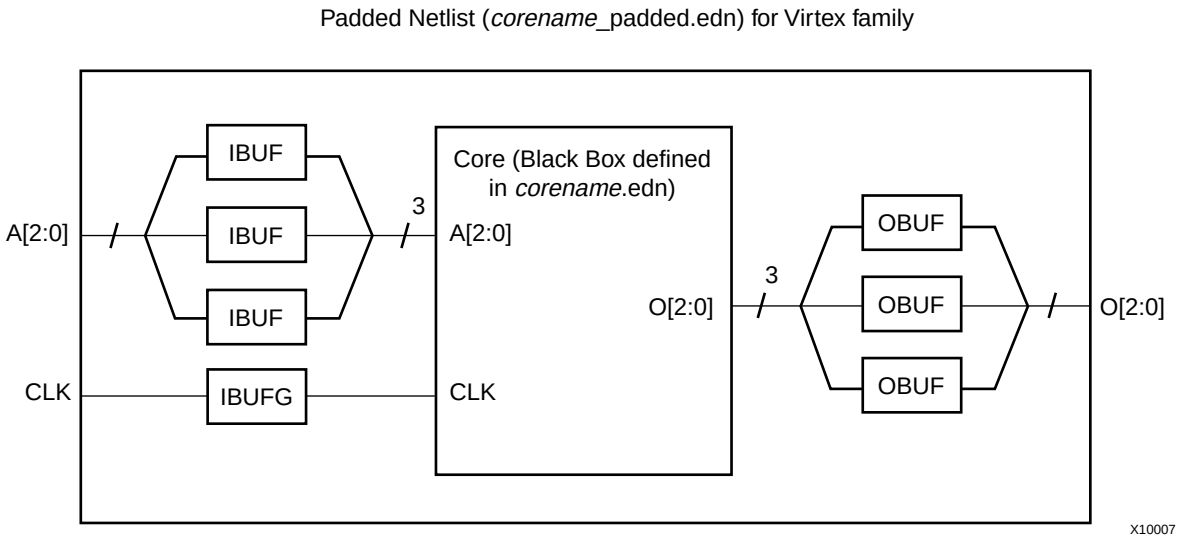


Figure 3-9: **Generate netlist wrapper with IO pads Example**

Padded files are generated for EDIF output only; they are not generated for VHDL and Verilog output.

- **Remove Placement Attributes**

  When selected, this option removes any RLOC and HU_SET attributes embedded in a parameterized core before writing out its EDIF netlist.

  The only EDIF netlists affected by this option are those that are generated by parameterized CORE Generator cores. EDIF netlists for fixed netlist cores are not affected by this option.

  Note that this option does not determine whether RLOCs and HU_SETs are generated when a core is elaborated. The Remove Placement Attributes option simply prevents RLOC and HU_SET values from being output to the EDIF netlist for a core by removing them if they are present.

  Some IP customization GUIs have a **Create RPMs** checkbox, which allows you to enable or disable the creation of RLOCs and HU_SETs in the core you are configuring. If the **Remove Placement Attributes** option is set, it overrides the **Create RPMs** check box, and RLOC and HU_SET constraints will not appear in the EDIF netlists of the cores for which **Create RPMs** was enabled.

- **Create NDF Synthesis Optimization Interface for NGC cores**

  In prior releases of the ISE software, CORE Generator created a single flat structural EDIF netlist with an .EDN extension for every IP core it generated. Third party synthesis tools (for example, Mentor Graphics™ LeonardoSpectrum™) used this EDN file to infer resources utilized by the core as well as rough timing information. The synthesis tool used this information to optimize the elaboration of the surrounding design logic.

Starting with CORE Generator™ 4.2i IP Update #2, the logic implementation of certain new CORE Generator IP is described by a combination of a top level EDN file plus one or more NGC files. For third party synthesis tools to infer resource utilization and timing from the NGC files associated with these new cores, you must enable the **Create NDF Synthesis Optimization Interface for NGC cores** project option to generate a new NDF format file for each NGC output file.

Refer to your synthesis tool documentation for information on support for this feature.

Netlist Bus Format

Sets the format in which bus signals are written in the output EDIF file. If you select **Other** as your design entry vendor, you will have to set a Netlist Bus Format.

Bus format can be specified (and will be written into the EDIF file) in either of these ways:

- As individual bus bits. For these options, B represents the name of the bus and I represents the bus index. Options are B<I>, B(I), B[I], and BI.
- As a single array. For these options, B represents the name of the bus and n:m represent the range of the bus index. Options are B<n:m>, B(n:m), and B[n:m].

Creating a Customized Core

To customize a core and add it to your Core Generator project:

1. Select the core in the Cores Catalog Browser.

   For a description of how to use the Cores Catalog Browser to find and select a core, see "Using the Cores Catalog Browser".

2. Select **Core** → **Customize** or click the Customize toolbar button.

   A core customization GUI appears for the selected core.

3. In the core customization GUI, set customization options for the core.

   Some notes about setting customization options:

   - For information about the customization options, click the **Data Sheet** button in the core customization GUI. The data sheet that appears explains all of the options.
   - In some of the customization GUIs, you can click on a pin in the GUI's symbol drawing to enable the pin and set options related to the pin. See "Setting Options Using the Core Symbol".
   - To view Solution Records related to this core, select the **Web Links** tab in the core customization GUI, then click **Solution Records for this Core**.

4. When you are finished setting customization options, click **Generate**.

   The core is elaborated and it appears in the Generated Modules window.

   Note: If you want to halt the elaboration before it is completed, click the **Cancel** button in the Generating Core message box.

If the CORE Generator™ System cannot generate all of the core's output products, a dialog box appears to indicate this fact. The dialog box contains a button allowing you to view error and warning messages related to the problem.

If the **Close IP Customization Dialog after Generation** preference is set (see "Setting Preferences"), the customization GUI will close automatically after the core is generated. If the preference is not set, the customization GUI will remain open. To close the GUI if it remains open, click **Dismiss**.

Recustomizing a Core

Often, it may be necessary to generate slightly different versions of the same core, or modify a previously generated core. The Recustomize option allows you to do this by preloading the parameter settings you originally specified when generating the core.

To recustomize a core:

1. In the Generated Modules window, select the core to recustomize.

   If you want to recustomize more than one core, you can make these extended selections:

   - To select consecutive cores, select the first core, then hold the **Shift** key while you select the last core.
   - To select cores that are not consecutive, select the first core, then hold the **Ctrl** key while you select each additional core.

2. Click the Recustomize toolbar button (to recustomize under current project settings)

OR

Select **Core** → **Recustomize** → **Under original project settings** or **Under current project settings**.

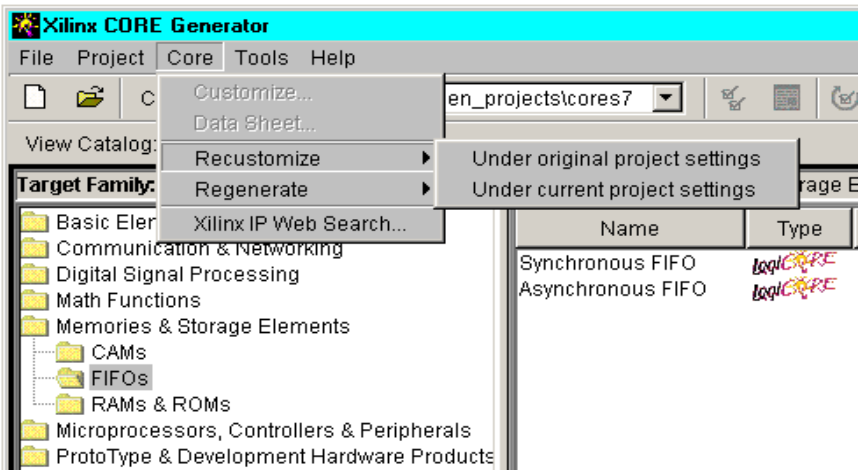


Figure 3-10: **Recustomize Menu**

You recustomize using one of these options:

- **Under original project settings** – The recustomized core is generated according to the project settings recorded in the core's XCO file when it was originally created, and the current project property settings are ignored.
- **Under current project settings** – The recustomized core is generated according to the current project settings.

3. In the core customization GUI for the first selected core, set customization options for the core.
4. When you are finished setting customization options, click **Generate**.

   The core is elaborated using the new customization settings.

   If the CORE Generator™ System cannot generate all of the core's output products, a dialog box appears to indicate this fact. The dialog box contains a button allowing you to view error and warning messages related to the problem.

   If the **Close IP Customization Dialog after Generation** preference is set (see "Setting Preferences"), the customization GUI will close automatically after the core is generated. If the preference is not set, the customization GUI will remain open. To close the GUI if it remains open, click **Dismiss**.

   If you selected more than one core to recustomize, a core customization GUI will appear for the next selected core as the customization GUI closes for the previous core.
5. If you have selected more than one core for recustomization, perform steps 3 and 4 for each additional core.

Regenerating a Core

There may be situations where you may wish to regenerate a core in your project, either to generate additional output products that were not previously specified or required, or to retarget your core to a different Xilinx™ architecture. When you choose to *regenerate* a core, the regenerated core is implemented using the *same* parameter values as specified previously, but is regenerated with the new Project Options settings (including target architecture).

To regenerate a core:

1. In the Generated Modules window, select the core to regenerate.

   If you want to regenerate more than one core, you can make these extended selections:

   - To select consecutive cores, select the first core, then hold the **Shift** key while you select the last core.
   - To select cores that are not consecutive, select the first core, then hold the **Ctrl** key while you select each additional core.
2. Click the Regenerate toolbar button (to regenerate under current project settings)

   OR

   Select **Core** → **Regenerate**. → **Under original project settings** or **Under current project settings**.

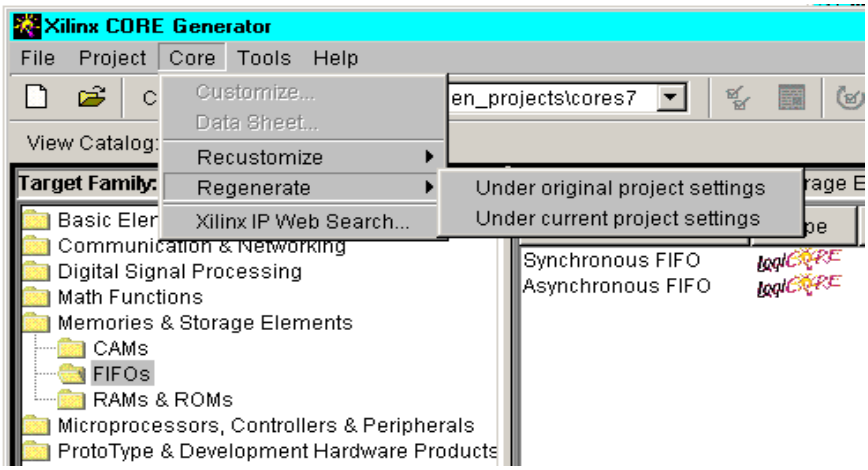


Figure 3-11: **Regenerate Menu**

You regenerate using one of these options:

- **Under original project settings** – The core is generated according to the project settings recorded in the core's XCO file when it was originally created, and the current project property settings are ignored.
- **Under current project settings** – The core is generated according to the current project settings.

The selected cores will be regenerated, one after another.

Selecting Target XILINX FPGA Family Options

The Xilinx CORE Generator™ System tailors your core 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. For example, cores that were targeted to the Spartan™ architecture when they were generated, will 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.

Using the Web Browser and the PDF Viewer

Another feature of the CORE Generator System is the ability to link to CORE Generator-related sites on the Web. The CORE Generator System can also open a PDF viewer to read core data sheets and user documents in PDF format.

Using links from the CORE Generator Help menu, you can click to these sites:

- The CORE Generator Examples page
- The Xilinx IP Center page
- The Xilinx Support and Services page
- The Xilinx home page

Refer to the following section on Setting Preferences for information on configuring CORE Generator to invoke your preferred web browser and PDF viewer applications.

Setting Preferences

Preferences are user-specific settings that apply globally to all of your projects. Preferences include the following things.

- The location of your web browser and PDF viewer executables
- Proxy settings (if a proxy is used)
- File overwrite behavior
- Whether to always automatically open the last project or not
- Cores Catalog filtering behavior
- Whether a core customization GUI will close or remain open after a core is customized.

To configure your CORE Generator™ preferences, select **File** → **Preferences**. The Preference Options dialog box appears.

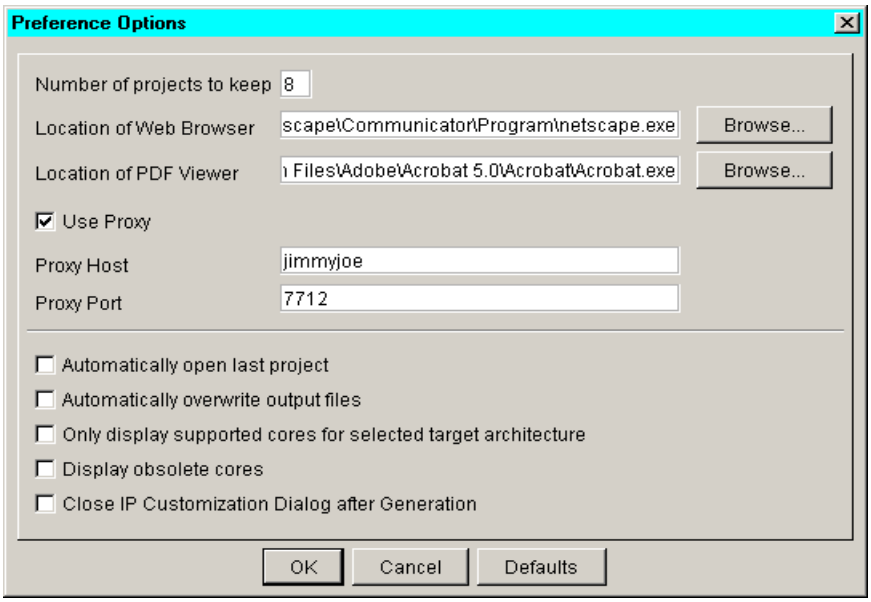


Figure 3-12: **Preference Options Dialog Box**

The following sections describe the preferences that you can set from the Preference Options dialog box.

Preferences are stored as follows:

- Windows™ – Preferences are stored in the Windows registry.
- UNIX™ workstations – Preferences are stored in your home directory in the .coregen.prf file.

Location of Web Browser

Specifies the location of your Netscape™ or Internet Explorer web browser.

Location of PDF Viewer

Specifies the location of your Acrobat™ or Netscape PDF viewer.

Use Proxy

If selected, allows you to specify a Proxy Host and Proxy Port. A network proxy is used to provide an additional level of security between your computer and the Internet. It is usually associated with a firewall. You may need to contact your system administrator for your required Proxy Host and Proxy Port settings.

Proxy Host and Proxy Port settings must be specified if you want to perform data transfers over the Internet from the CORE Generator IP Updates page.

Proxy Host and Proxy Port

If Use Proxy is selected, Proxy Host specifies the name of your proxy host and Proxy Port specifies the proxy port. Contact your system administrator for this information.

Automatically open last project

If selected, the CORE Generator™ System bypasses the Open Project dialog box at startup, and opens the last open project. If deselected, CORE Generator prompts you for a project at startup by displaying the Open Project dialog box.

Automatically overwrite output files

Sets the OverwriteFilesDefault preference, which defines the default behavior for all projects. This preference setting applies only if the similar, but project-specific OverwriteFiles property is set to Default.

Only display supported cores for target architecture

If selected, the Cores Catalog Browser will show only those cores that are supported by the target architecture setting for the current project.

Display obsolete cores

If selected, the Cores Catalog Browser display will include cores which are scheduled to be obsoleted. If deselected, these cores will not be displayed in the list.

Close IP Customization Dialog after Generation

If selected, the core customization GUI displayed for a core will close after the core is generated. If deselected, the core customization GUI will remain displayed after the core is generated.

CORE Generator Data Sheets

The Xilinx™ CORE Generator System provides all core data sheets in Adobe Acrobat PDF format.

Data sheets 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

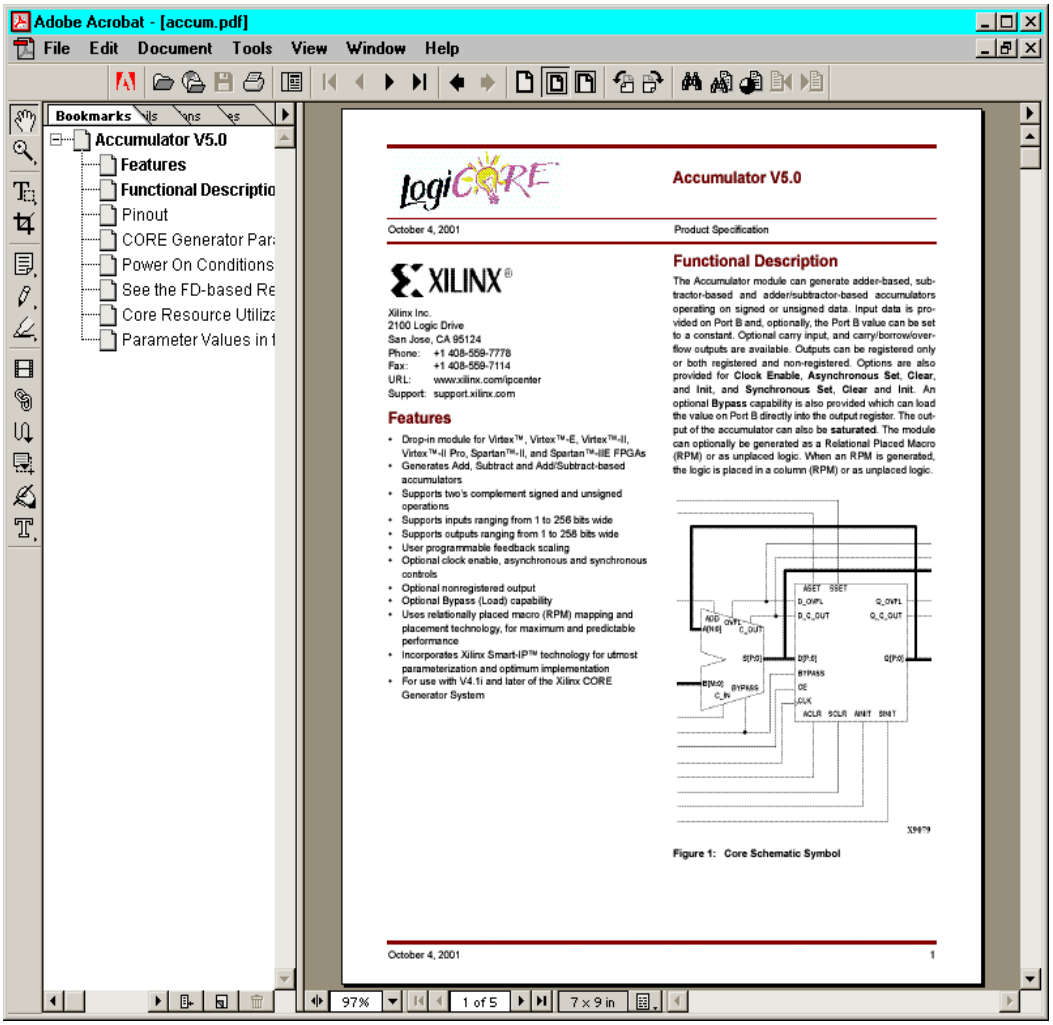


Figure 3-13: **Data Sheet**

You can view a core's data sheet in the following ways:

- By clicking on the **Data Sheet** button in the core's customization GUI
- By selecting the core in the Cores Catalog Browser and then selecting **Core** → **Data Sheet**.
- By selecting the core in the Cores Catalog Browser and then clicking the Data Sheet icon in the main CORE Generator toolbar.
- By right clicking the core name in the Cores Catalog Browser and clicking **Data Sheet** in the right-click menu.

This launches the Adobe Acrobat™ Reader and calls up the data sheet for the selected core.

Accessing Cores

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 in PDF format, which you can view using the Adobe Acrobat™ viewer (see "CORE Generator Data Sheets").

For each core, the CORE Generator System delivers the following:

- A customized EDIF netlist and, for some cores, one or more NGC netlists.
- Verilog or VHDL behavioral simulation wrapper files which map to parameterized simulation models in the XilinxCoreLib library
- Verilog or VHDL Instantiation templates
- ISE or Mentor schematic symbol support files.

Configuring the Cores Catalog Browser

The Cores Catalog Browser is a project-specific view of the IP customizers which are available to the current project. It can be configured to add (make visible) or remove (make invisible) any core customizers residing in the CORE Generator built-in repository ($XILINX / coregen / ip / xilinx).

Adding Core Customizers to the Cores Catalog

You are encouraged to always use the latest versions of all IP whenever possible so that you can have access to the latest features and bug fixes for every core. Ordinarily only the *latest* versions of all installed core customizers are displayed in the Cores Catalog Browser when you select All after installing an IP update. However under some conditions, as when it is necessary to minimize design changes, it may be necessary to modify an existing core using an older version of a core customizer. The older version of the core customizer can be accessed from Cores Catalog by performing a custom configuration of the catalog.

To make the older core customizer visible:

1. Select **Project** → **Cores Catalog Display** → **Customize**.
2. Double click the appropriate folders to find the core you wish to make visible in the Update Project Cores window. You may change the way the cores are organized by changing the setting in the View Catalog toolbar (notice that this window is very similar to the Cores Catalog Browser in the main CORE Generator GUI).
3. Click the checkbox next to the core so that it is selected. This will make it visible in the Cores Catalog for the project. For example, to make v5.0 of the Comparator core customizer visible in the Cores Catalog, select the Version 5.0 entry.

Visibility Example

1. Locate the core by navigating to **Math Functions** → **Comparators** as shown in Figure 3-14.
2. Click the checkbox next to the entry for **Comparator 5.0**.

3. When you return to the main CORE Generator™ GUI, this core is now visible in the catalog.

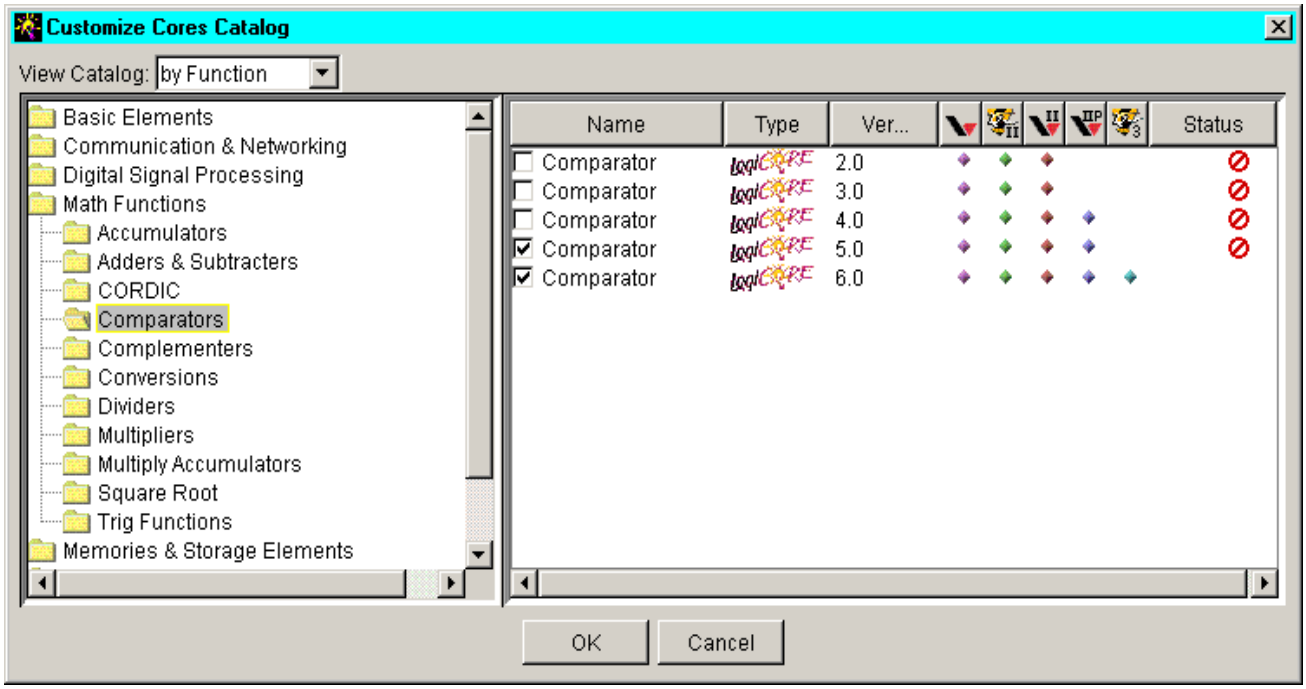


Figure 3-14: **Making the Comparator v5.0 Core Visible in the Cores Catalog Display**

Removing Cores from View in the Cores Catalog

The procedure for removing individual cores from view in the Cores Catalog Browser for your current project is similar to the procedure described above for making them visible. The only difference is that you must *uncheck* the core customizer's checkbox to remove it from view.

To remove a core from view in the Cores Catalog Browser:

1. Select **Project** → **Cores Catalog Display** → **Customize**.
2. Find the core customizer you wish to remove from view in the Customize Cores Catalog window that appears.
3. Click the checkbox next to the core so that it is no longer selected. When configured in this manner, the core customizer will not display in the Cores Catalog for the project.

Copying a Project

When you copy a project, you copy all of the cores from the project (the source project) to another project (the destination project). Cores from the source project are added to the existing cores in the destination project.

When you copy cores from one project to another, only the XCP files are copied. The XCP files describe the cores and how they are customized. Once the XCP files have been copied to the destination project, they can be regenerated within the destination project using its project options.

To copy a project:

1. Select **Project** → **Copy Project**.

   The Copy Project dialog box displays.

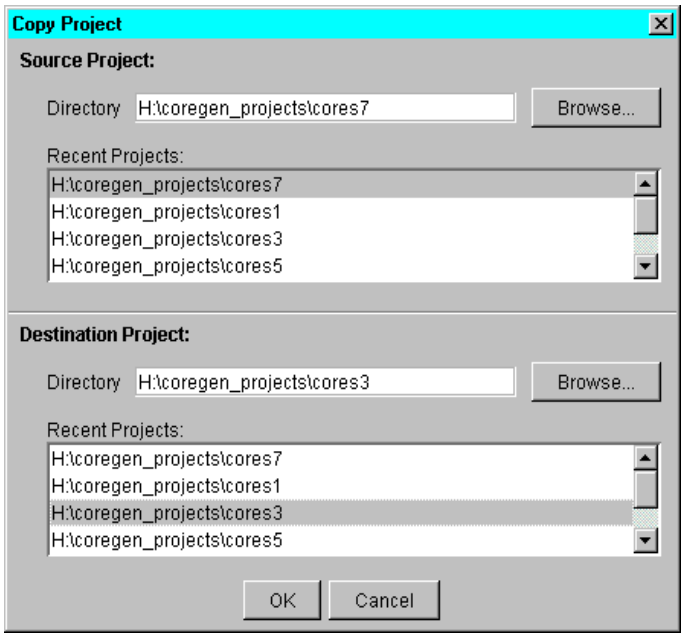


Figure 3-15: **Copy Project Dialog Box**

2. In the Source Project section of the Copy Project dialog box, specify the directory containing the project you want to copy in one of these ways.
   - Enter the directory name in the Directory box.

     OR
   - Browse to the directory using the Browse button.

     OR
   - Click one of the directories in the Recent Projects box.
3. In the Destination Project section of the Copy Project dialog box, specify the directory containing the project into which you want to copy the cores from the source project. Select the directory in one of these ways:
   - Enter the directory name in the Directory box.

     OR
   - Browse to the directory using the Browse button.

     OR
   - Click one of the directories in the Recent Projects box.
4. Click **OK**.

   The customized cores from the source project are copied into the destination project.

Input and Output Files

This section lists 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 or Name	Description
.CGF	As a log file, the CGF file is used in the Memory Editor to record the user-specified inputs that are used to generate the COE files for a memory (see .COE entry below). As a specification file, a CGF file can be used to define the data contents of COE files for memory blocks. For more details on the CGF file refer to "Memory Editor Overview" in Chapter 6.
.COE	ASCII input data file. Used when multiple data values must be specified for a core, usually as an array. Examples include specification of coefficient values for FIR Filters, mask patterns for Correlator cores, and initialization values for Memory modules and memory-based modules. See the $XILINX/coregen/data directory and "COE Files".
.XCO	CORE Generator input file containing the parameters used to regenerate a core. To use an XCO file as an input, select **File** → **Execute Command File** and specify the desired input XCO file. The XCO is also an output file generated by the CORE Generator. For more details on the XCO file refer to "XCO Files" in Chapter 4.
.XCP	An XCP file is basically an XCO file, minus the project specific settings such as FlowVendor, DesignFlow and miscellaneous output file formats. The purpose of the XCP file is to provide you with a data file which can be copied to another project to regenerate a core, using project settings that may be different from those in the original core project.

Table 3-6: **CORE Generator Output Files**

File Extension or Name	Description
.ASY	Graphical symbol information file. Used by the ISE tools and some third party interface 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.
*corename*_padded.edn	EDIF wrapper file generated for a core when the **Generate netlist wrapper with IO pads** project option is enabled. The file adds input pads and output pads to the core, allowing you to process the generated core through the Xilinx design flow as if it were a complete chip design.
*corename*_flist.txt	A text file listing all of the output files produced when a customized core was generated in the CORE Generator.
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 some CORE Generator modules when an HDL simulation flow is specified. A MIF data file is used to support HDL functional simulation of modules which use arrays of values. Examples include memories, FIR filters, and bit correlators.
.NDF	Optional output file produced for cores that generate NGC files. The NDF files allow third party synthesis tools to infer resource utilization and timing from the NGC files associated with these new cores.
.NGC	A binary Xilinx implementation netlist. Starting with CORE Generator 4.2i IP Update #2, the logic implementation of certain new CORE Generator IP is described by a combination of a top level EDN file plus one or more NGC files.
.V	Verilog wrapper file. File which is used to support Verilog™ functional simulation of a core. The V wrapper passes customized parameters to the Xilinx core. For more information, see “Verilog HDL Design Flow” in Chapter 5.
.VEO	Verilog template file. The components in this file can be used to instantiate a core. For more details refer to “Verilog HDL Design Flow” in Chapter 5.
verilog_analyze_order	This file lists the CORE Generator Verilog behavioral models in a suggested compiled order.

Table 3-6: **CORE Generator Output Files**

File Extension or Name	Description
.VHD	VHDL wrapper file. File which is used to support VHDL functional simulation of a core. The VHD wrapper passes customized parameters to the generic core simulation model. For more information, see "VHDL HDL Design Flow" in Chapter 5.
.VHO	VHDL Template file. The components in this file can be used to instantiate a core.
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.
.XCO	As an output file, the XCO file is a log file which records the settings used to generate a particular core. An XCO file is generated by the CORE Generator System for each core that it creates in the current project directory. For details on the XCO file refer to "XCO Files" in Chapter 4. An XCO file can also be used as an input to the CORE Generator. For information on how the XCO file can be used as an input, refer to the XCO file description in Table 3-5 above.
.XCP	As an output file, similar to the XCO file, except that it does not specify project-specific settings such as target architecture and output products.
.XSF	A port list information file used by the Mentor™ tools to create a symbol representing the core.
XilinxCoreLib/*.v	Verilog™ behavioral models extracted from the IP installed in the CORE Generator tree. The verilog /src/XilinxCoreLib directory is used as the source library for compiled simulators. Usually located in $XILINX/verilog/src by default.
XilinxCoreLib/*.vhd	VHDL behavioral models extracted from the IP installed in the CORE Generator tree. The vhdl /src/XilinxCoreLib directory is used as the source library for compiled simulators. Usually located in $XILINX/vhdl/src by default.
XilinxCoreLib/*_comp.vhd	VHDL component declaration files for each CORE Generator IP module extracted from the CORE Generator tree. The files are located in the $XILINX/vhdl/src directory by default.

Using Core Customization GUIs

The following sections describe details about using core customization GUIs to create customized cores. The topics discussed are:

Core Customization GUI Overview

There is a core customization GUI for each core included in the CORE Generator™ System. The GUI supplies information about the core and allows you to customize core parameters when you add a core to your project or recustomize the core afterwards.

Each core customization GUI contains four tabs:

- **Parameters** – Allows you to set the parameters through which you customize the operation of the core.
- **Core Overview** – Contains a functional description of the core and indicates the core version, source company, creation date, and type.
- **Contact** – Contains contact information for the company that developed the core.
- **Web Links** – Contains a listing of core-related pages on the Xilinx website, including solution records related to this core, pages related to the CORE Generator, and pages through which you can supply feedback to Xilinx.

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): a - z, 0 - 9, _(underscore)
- No extensions
- No HDL reserved words. Examples of words that cannot be used are:
  - Verilog keywords such as module, input, and output
  - VHDL keywords such as component, function, configuration, port, signal

Using Customization GUI Buttons

The following buttons are common to all customization GUIs:

- **Generate** – Assuming there are no conflicts with any of the specified parameters, clicking **Generate** causes the CORE Generator to create the requested files for the core.
- **Dismiss** – Closes the customization GUI and returns you to the Core Browser window without generating any files.
- **Data Sheet** – Invokes Adobe Acrobat™ to display the data sheet for the module being parameterized.
- **Version Info** – Lists New Features and Bug Fixes for this version of the module.

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 customization GUIs 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.

Using the Core Viewer

The Core Viewer shows a graphical representation of a core's footprint when it is implemented as a Relationally Placed Macro (RPM). This representation can be useful when you floorplan a large design. Only relationally placed logic is displayed because the mapping and placement of unconstrained (non-RPM) logic is indeterminate. If none of the logic for the core has been relationally placed, the Core Viewer simply indicates this with the message, `No RPMed Logic to Display`.

The Core Viewer also reports resource utilization and the percentage of the core's logic that has been relationally placed. The following figure illustrates what the Core Viewer might display for a relationally placed multiplier core

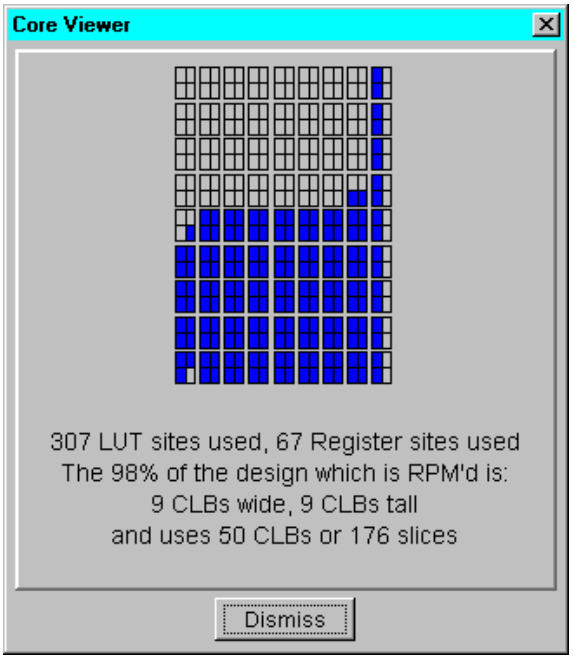


Figure 3-16: **Core Viewer Screen for a Multiplier Core**

Enabling the **Display Core Footprint** checkbox in a core customization GUI brings up the Core Viewer after the core has been generated. An additional **Create RPM** checkbox is provided in an IP core's customization GUI when the core can be generated either with or without RLOC constraints.

Setting Options Using the Core Symbol

In some of the new and newly-revised cores you can set configuration options in a core configuration GUI by selecting pins on the core's symbol drawing. Examples of these cores are the Content Addressable Memory (CAM) and Distributed Arithmetic FIR Filter. On the applicable symbols, the cursor will change to a hand when it passes over a selectable pin. As an example, the CORDIC customization GUI allows you to select the RDY symbol pin (see the following figure).

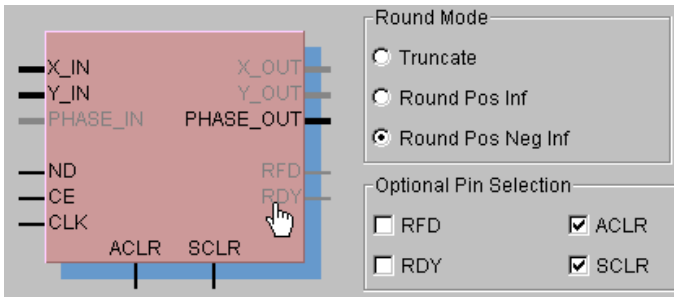


Figure 3-17: **Selecting a Symbol Pin in a Core Customization GUI**

When you select the pin, the symbol pin is activated and the **RDY** check box is selected to indicate the pin is available to signal that output data is ready (see the following figure).

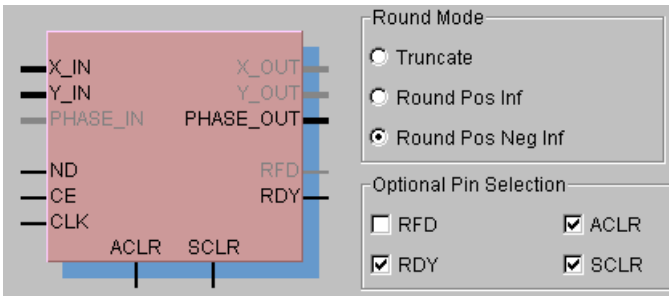


Figure 3-18: **Result of Pin Selection**

In some cases, selecting one pin will also select or deselect other pins in the symbol, because the functions of these pins are related to the function of the selected pin.

COE Files

Some cores, for example, FIR filters, RAM, Correlators and Memories, require specification of multiple data values such as coefficient, bit pattern, or initialization values. To specify the values for these modules, you must load a COE file using a **Load Coefficients** button in the core customization window. When you click the button, a browser dialog box appears for you to load a COE file.

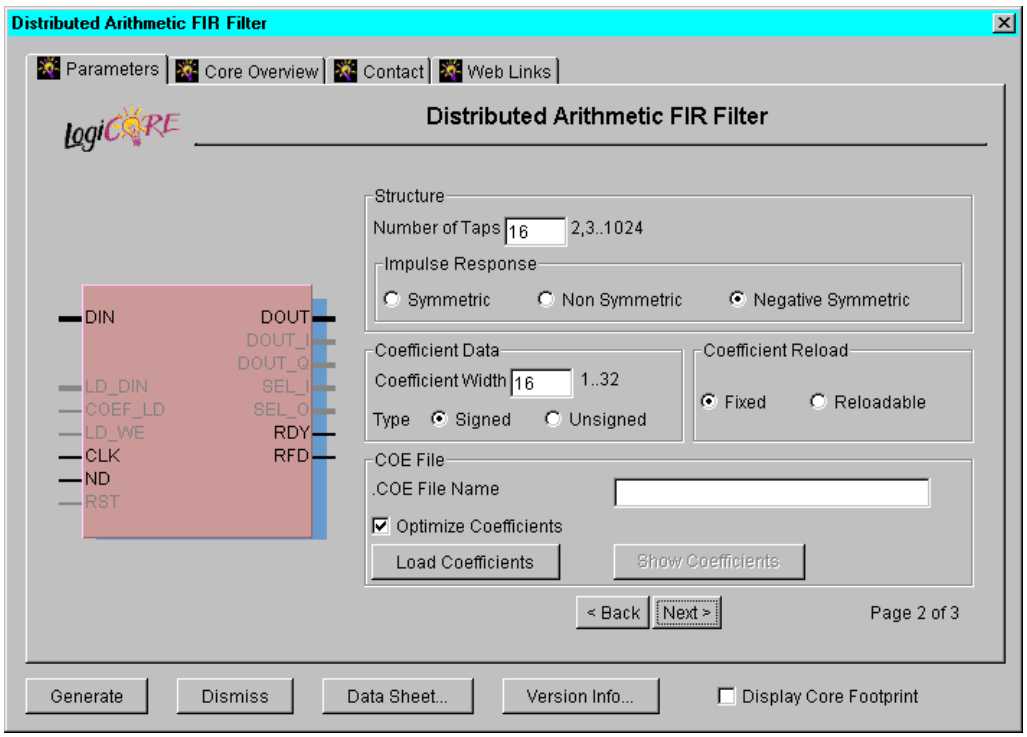


Figure 3-19: **DA FIR Filter Customization GUI**

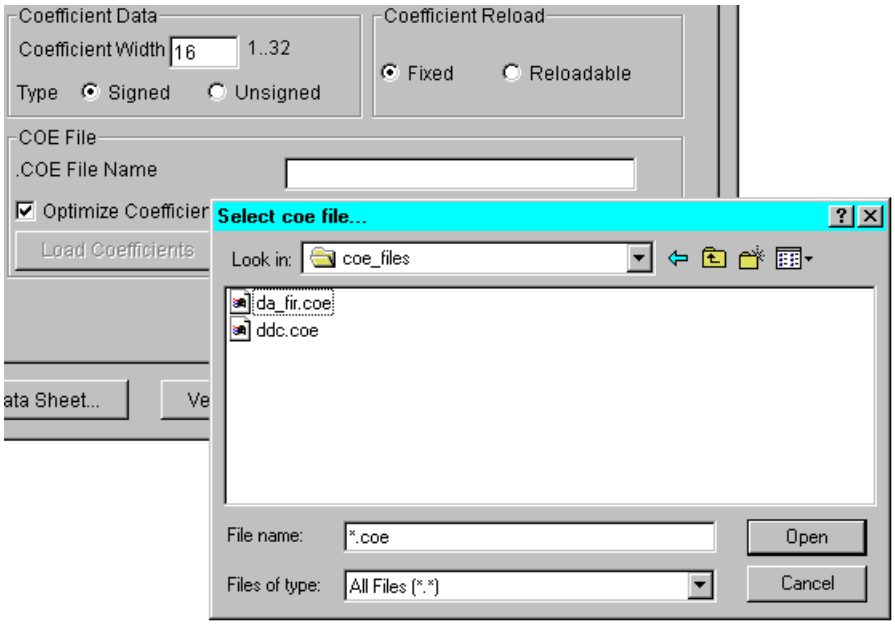


Figure 3-20: **Select COE File Dialog Box**

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
<Radix_Keyword> =Value ; Optional Comment

<Data_Keyword> =Data_Value1, Data_Value2, Data_Value3;
```

COE files for block and distributed memories can be created easily using the CORE Generator Memory Editor (see Chapter 6, "The Memory Editor").

The following table describes COE file keywords for specifying radix values for data. Keywords are not case sensitive. For information on the specific keywords required for a core, please refer to that core's data sheet.

Table 3-7: **Description of COE File Radix Keywords**

Keyword	Description
RADIX	Used for non-memory cores to indicate the radix being used to specify the coefficients of the filter.
MEMORY_INITIALIZATION_RADIX	Used for Virtex memory initialization values to specify the radix used.

The following table describes COE file keywords for data values. Keywords are not case sensitive.

Table 3-8: **Description of COE File Keywords for Data Values**

Keyword	Description
COEFDATA	Used for filters to indicate that the data that follows comprises the coefficients of the filter.
MEMORY_INITIALIZATION_VECTOR	Used for Virtex™ block and distributed memories.
PATTERN	Used for Bit Correlator COE files
BRANCH_LENGTH_VECTOR	Used in Interleaver COE files
MEMDATA	Obsolete keyword. Was used for XC4000 distributed memories.

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

One of the following keywords must be the last keyword specified in the COE file:

- COEFDATA
- MEMORY_INITIALIZATION_VECTOR
- MEMDATA (an obsolete keyword, supported for backward compatibility)

Any other keywords that follow are ignored.

You can find examples of COE files for the Distributed Arithmetic FIR Filter, Bit Correlator, Digital Down Converter, RAM-based Shift Register, Distributed Memory, and Virtex block RAM cores in the $XILINX/coregen/data directory.

Below is a selection of the COE files from this area.

```
************************************************************
*************  Example of Virtex Bit Correlator.COE  *************
************************************************************
; Sample .COE coefficient file for v2.0 and later
; versions of the Bit Correlator core.
;
; In this core, a COE file is used to specify the value
; of the bit mask when the Pattern Mask option is selected.
;
; Specifications:
;
; - 19 taps, hexadecimal coefficients
; - Serial input data
;
; Please refer to the datasheet for this core for more
; details on using the Mask option.

radix = 16;
pattern = 3 0 3 1 0 1 1 3 0 2 2 2 3 0 1 1 3 0 3;


************************************************************
*********  Example of Dual Port Block Memory .COE file  **********
************************************************************
; Sample memory initialization file for Dual Port Block Memory,
; v3.0 or later.
;
; This .COE file specifies the contents for a block memory
; of depth=16, and width=4.  In this case, values are specified
; in hexadecimal format.

memory_initialization_radix=2;
memory_initialization_vector=
1111,
1111,
1111,
1111,
1111,
0000,
0101,
0011,
0000,
1111,
1111,
1111,
1111,
1111,
1111,
1111;

************************************************************
********  Example of Single Port Block Memory .COE file  *********
************************************************************
; Sample memory initialization file for Single Port Block Memory,
; v3.0 or later.
;
; This .COE file specifies initialization values for a block
; memory of depth=16, and width=8. In this case, values are
; specified in hexadecimal format.
```

```
memory_initialization_radix=16;
memory_initialization_vector=
ff,
ab,
f0,
11,
11,
00,
01,
aa,
bb,
cc,
dd,
ef,
ee,
ff,
00,
ff;


****************************************************************
************  Example of Distributed Memory .COE file  ************
****************************************************************
; Sample memory initialization file for Distributed Memory v2.0 and
; later.
;
; This .COE file is NOT compatible with v1.0 of Distributed Memory Core.
;
; The example specifies initialization values for a memory of depth= 32,
; and width=16. In this case, values are specified in hexadecimal
; format.

memory_initialization_radix = 16;
memory_initialization_vector = 23f4 0721 11ff ABe1 0001 1 0A 0
 23f4 0721 11ff ABe1 0001 1 0A 0
 23f4 721 11ff ABe1 0001 1 A 0
 23f4 721 11ff ABe1 0001 1 A 0;


****************************************************************
******  Example of Distributed Arithmetic FIR Filter .COE file ***
****************************************************************
; Example of a Distributed Arithmetic (DA) FIR Filter .COE file
; with hex coefficients, 8 symmetrical taps, and 12-bit
; coefficients.
;
; Compatible with all versions of the Distributed Arithmetic
; FIR Filter which supports Virtex/Virtex-II and Spartan-II

Radix = 16;
CoefData= 346, EDA, 0D6, F91, F91, 0D6, EDA, 346;
```

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. An XCO file can also be edited and renamed to generate a slightly different core.

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

For a complete description of the CORE Generator batch interface, see "Batch Mode" in Chapter 4.

Syntax

```
coregen [ -i path_to_coregen_ini_file_name ]
[ -p project_path ] [ -q polling_dir_path ]
[ -intstyle <ise|xflow|silent> ] [-h]
-b module_name.xco
```

Performing CORE Generator Operations in Xilinx ISE

A number of CORE Generator operations can be performed within the Project Navigator of the Xilinx™ Integrated Software Environment (ISE). Most of the operations are performed without opening the CORE Generator window.

Within the Project Navigator, you can perform these CORE Generator procedures:

- Adding a customized core to a Project Navigator project
- Recustomizing a core
- Regenerating a core
- Regenerating all of the cores in a project
- Viewing the HDL functional model for a core
- Opening the CORE Generator window to manage the cores in a project
- Viewing the CORE Generator log

All of these operations are described in the *ISE Guide*, the ISE online help system. The core related help topics are grouped under **FPGA Design → Using Intellectual Property (Cores)** in the help Table of Contents.

Integrating CORE Generator into Applications

The CORE Generator provides a number of interfaces for integration into other applications. Polling mode allows an application to communicate with the CORE Generator through files. The polling interfaces provided by the CORE Generator, which allow you to integrate it into other applications, are described in "Polling Mode" in Chapter 4.

ASY and XSF Symbol Information Files

The ASY and XSF files are produced by the CORE Generator as symbol information files for various third party EDA tools.

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

The XSF file is a Xilinx Netlist Format (XNF) portlist file used by the Mentor tools to create a symbol representing the core.

Chapter 4

Batch Mode and Polling Mode

This chapter describes how CORE Generator™ operates in batch mode and in polling mode. The chapter includes the following sections:

Batch Mode

In batch mode, the CORE Generator runs the commands in a command file. The commands must be valid CORE Generator commands. You can run CORE Generator in batch mode in the following ways:

- When you invoke the CORE Generator from the command line. The command line options to invoke the CORE Generator in batch mode are described in "Command Line Options" in Chapter 2.
- By running a command file from the CORE Generator window. In the CORE Generator window, select **File** → **Execute Command File** and specify the desired input file. You may generate a core in batch mode by specifying an XCO file as the input file.

Batch Mode Command Line Options

The options to the **`coregen`** command that invoke the CORE Generator in batch mode are described in "Command Line Options" in Chapter 2.

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 most 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. See "Command Line Options" in Chapter 2 for more information.

The five types of command files in the CORE Generator™ are:

- coregen.ini / coregen_*user_name*.ini files
- user-generated command files
- XCO files
- XCP files
- coregen.log files

coregen.ini/coregen_*user_name*.ini

The CORE Generator loads and executes INI files when it is first invoked and also 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 you open a CORE Generator 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 information about the commands that may appear in a command file, see "CORE Generator Commands".

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. This file should not be edited in most cases. The related XCP files serve a similar function and can be edited.

As a log file, XCO files can be used to verify the settings for all the options that were used to generate a core. It can also be used to recreate the core exactly if you specify it as the command file input to **File** → **Execute Command File** from the CORE Generator main menu bar, or if you specify it as the argument to the **-b** option in batch mode (see "Command Line Options" in Chapter 2 for more details). In addition, double-clicking a core in the Generated Cores panel of the main CORE Generator GUI loads the customization parameter settings specified in the corresponding XCO file into its Customization GUI, allowing you to recustomize the core. This is useful when a core needs to be regenerated with current project settings (HDL outputs, target architecture, etc.), but with some parameter changes. See "Recustomizing a Core" in Chapter 3 for more details on this feature.

XCO File Syntax

Comment lines begin with the # character. Any output format or options lines start with the keyword SET. 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 XCO file for a version 7.0 Distributed Arithmetic FIR Filter generated using the dafir.coe example listed in "COE Files" in Chapter 3.

```
# Xilinx CORE Generator 6.1i
# Username = bob
# COREGenPath = C:\xilinx\coregen
# ProjectPath = H:\coregen_projects\cores7
# ExpandedProjectPath = H:\coregen_projects\cores7
# OverwriteFiles = False
# Core name: dafir
# Number of Primitives in design: 173
# Number of CLBs used in design cannot be determined when there is no
# RPMed logic
# Number of Slices used in design cannot be determined when there is no
# RPMed logic
# Number of LUT sites used in design: 62
# Number of LUTs used in design: 50
# Number of REG used in design: 72
# Number of SRL16s used in design: 12
# Number of Distributed RAM primitives used in design: 0
# Number of Block Memories used in design: 0
# Number of Dedicated Multipliers used in design: 0
# Number of HU_SETs used: 0
#
SET BusFormat = BusFormatAngleBracket
SET XilinxFamily = Virtex
SET OutputOption = OutputProducts
SET FlowVendor = Synopsys
SET FormalVerification = None
SET OutputProducts = ImpNetlist BmmFile
SELECT Distributed_Arithmetic_FIR_Filter Virtex Xilinx,_Inc. 7.0
CSET impulse_response = Symmetric
CSET number_of_channels = 1
CSET clock_cycles_per_output_sample = 11
CSET optimize_coefficients = true
CSET coefficient_reload = Fixed_Coefficients
CSET register_output = false
CSET coefficient_data_type = Signed
CSET coefficient_width = 12
CSET input_data_type = Signed
CSET implementation_option = Clock_Cycles_Per_Output_Sample
CSET component_name = dafir
CSET sample_rate_change = 1
CSET zero_packing_factor = 1
CSET input_data_width = 10
CSET number_of_taps = 8
CSET filter_type = Single_Rate_FIR
CSET coefficient_file = C:\coe_files\da_fir.coe
CSET reset = false
GENERATE
```

XCP files

An XCP file is an XCO file minus all “SET” commands, which are project-specific. These include the commands that specify Flow Vendor, DesignFlow and miscellaneous output file formats. The purpose of the XCP file is to provide you with a way to port a previously generated core to different projects and allow it to be regenerated using the new project’s settings.

The XCP file corresponding to the XCO file listed in the previous section looks like the following:

```
# Xilinx CORE Generator 6.1i
SELECT Distributed_Arithmetic_FIR_Filter Virtex Xilinx,_Inc. 7.0
CSET impulse_response = Symmetric
CSET number_of_channels = 1
CSET clock_cycles_per_output_sample = 11
CSET optimize_coefficients = true
CSET coefficient_reload = Fixed_Coefficients
CSET register_output = false
CSET coefficient_data_type = Signed
CSET coefficient_width = 12
CSET input_data_type = Signed
CSET implementation_option = Clock_Cycles_Per_Output_Sample
CSET component_name = dafir
CSET sample_rate_change = 1
CSET zero_packing_factor = 1
CSET input_data_width = 10
CSET number_of_taps = 8
CSET filter_type = Single_Rate_FIR
CSET coefficient_file = C:\coe_files\da_fir.coe
CSET reset = false
GENERATE
```

coregen.log

The coregen.log file is a log file that is automatically written by the CORE Generator. The file contains all the actions performed and messages displayed during a CORE Generator session. You can refer to the log to see what occurred during that session. A coregen.log file cannot be replayed to recreate a session.

- Windows™ – The coregen.log file is written to the directory where the **Start in** field in the Windows shortcut for the CORE Generator executable points. Normally this is the directory %XILINX%\bin\nt. If the CORE Generator is run in batch or polling mode, coregen.log is written to the directory where coregen.bat is invoked. If that directory is not writeable, coregen.log is written to your home directory.
- UNIX™ Workstation – The coregen.log file is written to the directory from which the coregen executable is invoked. If that directory is not writeable, coregen.log is written to your home directory.

CORE Generator Commands

The following table describes the CORE Generator commands, command arguments, and command functions that can be used in CORE Generator command files.

Table 4-1: **CORE Generator Commands**

Command	Arguments	Function
COPYXCPFILES	<*src_project_path*> <*destination_project_path*>	Copies all the XCP files from one project to another.
CSET	<*core_property=value*>	Sets a core property value.
END	N / A	Terminates the CORE Generator session.
EXECUTE	<*command_file_path*>	Executes the indicated command file.
GENERATE	N / A	Elaborates the currently selected core.
LAUNCHXCO	<*xco_filename*>	Opens a core customization GUI for a specific core, preloaded with the parameter settings saved in the specified XCO file. Uses the project parameters specified in the XCO file.
LAUNCHXCP	<*xcp_filename*>	Opens a core customization GUI for the specific CORE, preloaded with the parameter settings specified in the XCP file. Uses project parameters specified in the current project.
LOCKPROPS		(Obsolete)
NEWPROJECT	<*project_path*>	Creates a new project in the specified directory.
REGENERATEALLCORES	N / A	Regenerates all cores in the current project using current project settings. The XCO files in the current project are used as inputs for regenerating the cores.
SELECT	<*core_name*> <*architecture*> <*vendor*> <*core_version*>	Selects the indicated core.

Table 4-1: **CORE Generator Commands**

Command	Arguments	Function
SET	<*global_property=value*> <*project_property=value*>	Sets a CORE Generator™ property value.
SETPROJECT	<*project_path*>	Changes the current project to the specified property value.
UNLOCKALLPROPS		(Obsolete)
UNLOCKPROPS		(Obsolete)

Supported Commands in XCO and XCP Files

The following table describes the actions the CORE Generator will perform in batch mode when it encounters commands in an XCO or XCP file.

Table 4-2: **Commands in XCO and XCP Files**

Command	In XCO File	In XCP File
LOCKPROPS, UNLOCKPROPS, UNLOCKALLPROPS[a]	Issue warning and ignore	Issue warning and ignore
CSET, SELECT	Execute command	Execute command
SET	Execute command	Ignore with no warning
SET LockProjectProps[a]	Issue warning and ignore	Ignore with no warning
SET OverwriteFiles	Ignore with no warning	Ignore with no warning
GENERATE Note: There should be only one GENERATE command per XCO or XCP file. If more than one exists only the first one is executed and the rest ignored. You can only generate one core per XCO or XCP file.	Execute command	Execute command
All other commands, including CORESELECT	Ignore with no warning	Ignore with no warning

a. Starting in the next CORE Generator release the LOCKPROPS, UNLOCKPROPS, UNLOCKALLPROPS, and SET LockProjectProps commands will no longer be supported.

CORE Generator Global Properties

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

Table 4-3: **Global Properties**

Global Properties	Values	Description
CoreGenPath	*path*	Specifies the path to the *coregen* install directory.
CoreSelect	SpecifiedVersion | LatestVersion	This property applies only when reading an XCO file in batch mode. SpecifiedVersion: Uses only the core version specified. LatestVersion: Uses the latest version of the specified core which is available in the repository.
LockProjectProps[a]	true | false	Set this global property to true to lock the project and thus prevent other users from simultaneously accessing the project. To unlock the project, reset this property to false.
ProjectOverride	true | false	Specify true to use the current project's attributes instead of those listed in the XCO file. Specify false to use the attributes in the XCO files.
Username	<*username*>	This is your login name.

a. The LockProjectProps global property will no longer be supported, starting in the next CORE Generator release.

Project Properties

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

Table 4-4: **Project Properties**

Command	Values	Description
BusFormat	BusFormatAngleBracket | BusFormatSquareBracket | BusFormatParen | BusFormatNoDelimiter | BusFormatAngleBracketNotRipped | BusFormatParenNotRipped | BusFormatSquareBracketNotRipped	Sets the indicated output bus formatting.
DesignFlow	Schematic | VHDL | Verilog	Schematic: Directs the CORE Generator to create a symbol information file for the selected vendor (XSF or ASY) for each core it generates. VHDL: Directs the CORE Generator™ to create a corresponding VHO HDL template and VHD wrapper file to be used in conjunction with the static parameterized behavioral HDL models in $XILINX/Vhdl/src/XilinxCoreLib. Verilog™: Directs the CORE Generator to create a corresponding VEO HDL template and V wrapper file for use with the static behavioral HDL models in $XILINX/verilog/src/XilinxCoreLib.
ExpandedProjectPath	*project_path*	Specifies the expanded path to the CORE Generator install directory.
FlowVendor	Foundation_iSE | Innoveda | MentorSchematic | MentorHDL | Synplicity | Synopsys | Cadence | Other	Signifies which vendor toolset you have selected to simulate and develop the core design.
FormalVerification	Formality | Verplex | None	Selects one of the two supported formal verification vendors. None is the default.

Table 4-4: **Project Properties**

Command	Values	Description
OutputOption	DesignFlow | OutputProducts	Selects the project output options menu configuration style. DesignFlow (the default) assumes a primary vendor has been selected to support the overall CAE flow. OutputProducts allows you to select individual core elaboration outputs and formats.
OverwriteFiles	True | False | Default	True allows the existing output files to be overwritten. Default means the overwrite behavior is determined by the value of the OverwriteFileDefault preference.
ProjectPath	*<project_path>*	Specifies the path to the current project.
SimElabOptions	AddPads RemoveRPMs CreateNdf None	Core elaboration options that apply to parameterizable cores only. AddPads results in pads being added to I/O and clk pins in the core's EDIF netlist. RemoveRPMs results in the placement directives being removed from the core's EDIF netlist. CreateNdf generates an NDF file when an NGC core is elaborated. None is default.
SimulationOutputProducts	VHDL | Verilog	Specifies the output products to be created for elaboration to support HDL simulation.
XilinxFamily	Spartan2| Spartan3 | Virtex | Virtex2 | Virtex2P	Target Xilinx™ architecture for the CORE Generator project. Only one target architecture is allowed per project.

Polling Mode

The CORE Generator™ can be invoked in polling mode, which on the surface looks the same as the standard GUI mode. Polling mode allows an application to communicate with the 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** *poll_dir_path* option 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.

- The first line contains the following:
  - User assigned core name
  - Name of the core
  - Core version
- The second line contains the keyword SUCCESS or ERROR depending on whether the core was successfully generated. An example of the contents of a coregen.fin file is:

```
regaddr Registered_Addr 1.0
SUCCESS
```

When an application finds the keywords SUCCESS or ERROR in the coregen.fin file, the application can review the various log files to determine the appropriate processing for the core. 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. If a coregen.fin file remains, the CORE Generator overwrites it 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 coregen.cmd. This file 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 coregen.cmd file. When it detects the keyword COMPLETE, CORE Generator sequentially executes the commands in the file. After completing the last command in the coregen.cmd file, CORE Generator deletes the file.

Chapter 5

Schematic and HDL Design Flows

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

Understanding Schematic Design Flows

The CORE Generator System produces an EDIF Implementation Netlist (EDN) for schematic design flows. It may also produce NGC files for some cores. For ISE and Mentor™ flows, the CORE Generator also produces schematic symbol information files. The EDN file and NGC files contain information for implementing the module. The symbol files allow you to integrate the module into a schematic for ISE's schematic editor (ECS) and for Mentor tools.

ISE Design Flow

For details on how to integrate CORE Generator modules into an ISE design, refer to the *ISE Guide*, the ISE online help system. The core related help topics are grouped under **FPGA Design** → **Using Intellectual Property (Cores)** in the *ISE Guide* Table of Contents.

Mentor Design Flows

Mentor eProduct (Formerly Innoveda) Design Flow

Cores which are to be integrated into an eProduct™ schematic design flow should be generated using the Xilinx/eProduct schematic interface tool (ePDCore). This tool is invoked from within the eProduct schematic editor (Viewdraw™).

For more details on the eProduct Xilinx flow, see Answer Record #11683 at the Xilinx Answers Search.

Mentor Design Architect Flow

The CORE Generator System is integrated into the Mentor™ Design Architect™ schematic editor. Please refer to the *Mentor Interface Guide* documentation for details on integrating your CORE Generator module into a Design Architect schematic design.

Cadence Design Flow

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

For further information about how to integrate a core into a Concept-HDL schematic, please refer to the Xilinx answer here:

http://support.xilinx.com/techdocs/2005.htm

Introduction to HDL Design Flows

Increasing design size and complexity, as well as recent improvements in design synthesis and simulation tools, have made HDL the preferred design language of most integrated circuit designers. The two leading HDL synthesis and simulation languages today are Verilog™ and VHDL. To integrate a CORE Generator™ module into an HDL design, refer to the design flow in the following figure.

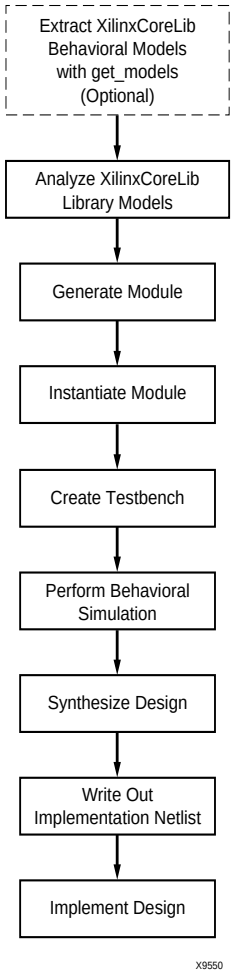


Figure 5-1: **HDL Front End Design Flow Chart**

HDL Behavioral Simulation Flow Features

The CORE Generator HDL behavioral simulation flow features a parameterized behavioral simulation model library (XilinxCoreLib), VHDL / Verilog wrapper files, and instantiation template files.

A more comprehensive list of the features is as follows:

- XilinxCoreLib Simulation Library

  The CORE Generator System provides both Verilog and VHDL `XilinxCoreLib` behavioral simulation libraries to support functional simulation of the CORE Generator cores.

The `XilinxCoreLib` libraries are provided in source file format at standard locations in the Xilinx™ installation tree.

The Verilog™ library is located here:

$XILINX / verilog / src / XilinxCoreLib,

and the VHDL library is located here:

$XILINX / vhdl / src / XilinxCoreLib.

The libraries are automatically updated when you install Core IP updates.

- compxlib (Compile Xilinx HDL Libraries)

  Automated support for compiling all Xilinx HDL libraries (including XilinxCoreLib for CORE Generator IP) is integrated into the Xilinx application "compxlib". Please consult the *Synthesis and Verification Design Guide* for detailed information on how to use this application.

- get_models Utility

  Utility for extracting HDL simulation models from an IP repository to a single library. The get_models utility also generates analyze order information for the CORE Generator Verilog and VHDL libraries which is needed when compiling these models with compiled HDL simulators.

  The get_models utility is available by selecting **Tools** → **Get Models**. This utility is an advanced feature that does not need to be run in most cases since all IP models are either automatically extracted during the install process, or shipped pre-extracted. See "GetModels Overview" in Appendix A for details on get_models.

- coredb Utility

  The coredb command line utility updates or regenerates the primary.mnf installed IP manifest file located in $XILINX / coregen / ip / xilinx / primary. The MNF file contains a summary of all cores installed in the CORE Generator built-in repository. In normal usage it is usually not necessary to run coredb since it executes automatically during the CORE Generator startup process after you install an IP module update. The coredb utility can be run manually if needed to explicitly force an update of the primary.mnf file if the primary.mnf file becomes corrupted due to an install problem.

  Syntax: **`coredb`**

- HDL Wrapper Files

  The CORE Generator creates either a Verilog or VHDL wrapper file when HDL output products are requested. The Verilog wrapper file name is module_name.v, and the VHDL wrapper file name is module_name.vhd. The wrapper files are used to deliver the customization constructs to the corresponding parameterized XilinxCoreLib simulation models, providing a simpler user interface for behavioral simulation. The wrapper files must be analyzed the same way actual behavioral simulation models would be when performing functional simulation.

- Instantiation Template Files

  The VEO and VHO files are generated by the CORE Generator to be used as HDL instantiation templates. Starting with the 4.1i release, the VEO and VHO files only contain instantiation template code snippets. The parameter passing and configuration code snippets which were written to these templates in earlier CORE Generator releases are now written out to the wrapper files instead.

- verilog_analyze_order File

  This file lists the CORE Generator Verilog™ behavioral models in a suggested compiled order. It may be updated by running Get Models from the CORE Generator™ **Tools** menu.

  Starting with the 4.1i release, XilinxCoreLib Verilog simulation models are compile order independent. The verilog_analyze_order file continues to be provided to ensure continuity of support for any designers that may be using this file as an input to a user compile script.

- vhdl_analyze_order File

  This file lists the CORE Generator VHDL behavioral models in the order in which they must be compiled for simulation. It may be updated by running Get Models from the CORE Generator Tools menu.

  More than one compile order may be valid for compilation of the VHDL XilinxCoreLib library.

Creating Verilog Designs

This section briefly describes the procedure for behavioral simulation, synthesis, and implementation of Verilog™ designs containing CORE Generator™ modules. To integrate a CORE Generator module into an HDL design, refer to the design flow in the following figure.

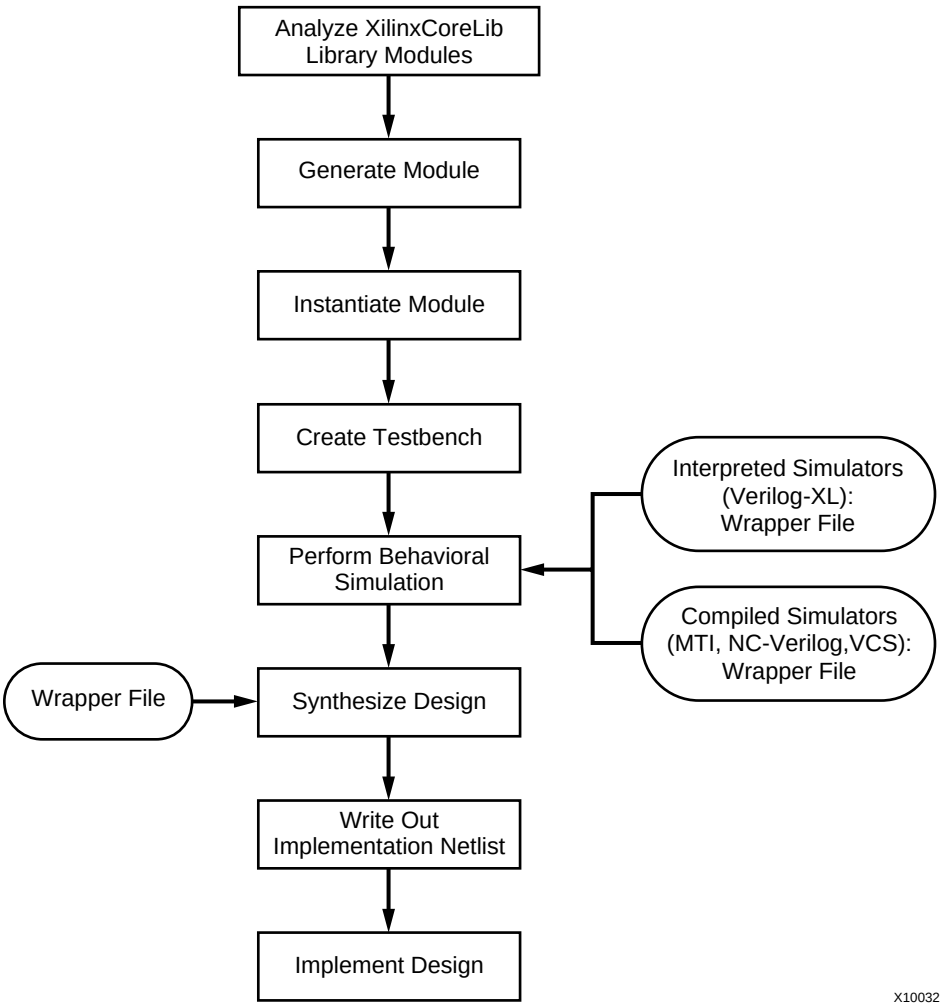


Figure 5-2: **Verilog Front End Design Flow Chart**

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 vendor tools:

Table 5-1: **Vendor Tools for Verilog Flow**

Function	Tools
Synthesis	Synopsys™ FPGA Compiler II™ Synplicity Synplify™ Mentor Graphics™ LeonardoSpectrum™ Xilinx™ XST
Simulation	MTI ModelSim™ / VLOG Cadence™ Verilog-XL Cadence NC-Verilog Synopsys VCS / VCSi

Verilog Design Flow Procedure

This section describes the detailed procedure for the Verilog design flow.

Verilog source format simulation models for the CORE Generator cores are provided in $XILINX / verilog / src / XilinxCoreLib. Compile the XilinxCoreLib library using one of the following procedures.

1. Use the Xilinx compxlib library compilation script to compile the XilinxCoreLib library source models. The script is located at $XILINX / bin / <*platform*> / . For details on how to use this script, refer to the information on the compxlib compile script in "HDL Behavioral Simulation Flow Features".

   OR

   Compile XilinxCoreLib using the individual commands listed in the following table.

Table 5-2: **Simulation Instructions**

Vendor	Setup File Settings	Commands
MTI	None	vlib <*dir_to_compiled_XilinxCoreLib*> vmap XilinxCoreLib <*dir_to_compiled_XilinxCoreLib*> vlog +incdir+$XILINX/verilog/src -work XilinxCoreLib $XILINX/verilog/src/glbl.v {files in verilog_analyze_order}
ncverilog	Add the following in cds.lib: DEFINE XilinxCoreLib {*compiled_XilinxCoreLib*} Add the following in hdl.var: DEFINE LIB_MAP ($LIB_MAP, $XILINX/verilog/src/XilinxCoreLib=>XilinxCoreLib) DEFINE VIEW_MAP ($VIEW_MAP, .v => v)	ncvlog -MESSAGES -CDSLIB cds.lib -HDLVAR hdl.var -INCDIR $XILINX/verilog/src -WORK XilinxCoreLib $XILINX/verilog/src/glbl.v {files in verilog_analyze_order} ncpack -MESSAGES -CDSLIB cds.lib XilinxCoreLib
vcs/vcsi	None	vcs -q -Mupdate -Mdir=<*compiled_XilinxCoreLib*> -Mlib=<*compiled_XilinxCoreLib*> -y <*src_XilinxCoreLib*> +libext+.v +incdir+$XILINX/verilog/src $XILINX/verilog/src/glbl.v {files in verilog_analyze_order}
Verilog-XL	N/A	N/A

2. In the CORE Generator™, select **Project** → **Project Options**. Click **Flow Vendor** in the Output Options panel. The Design Entry panel appears in the right hand side of the dialog. From the Design Entry panel, select **Verilog** in the first column and your target Vendor in the second column. This specifies the vendor software you will be using for your Verilog design, and accordingly, the preferred EDIF bus delimiter format for compatibility with the synthesis tool output. This ensures proper integration with the upper level parent implementation netlist.

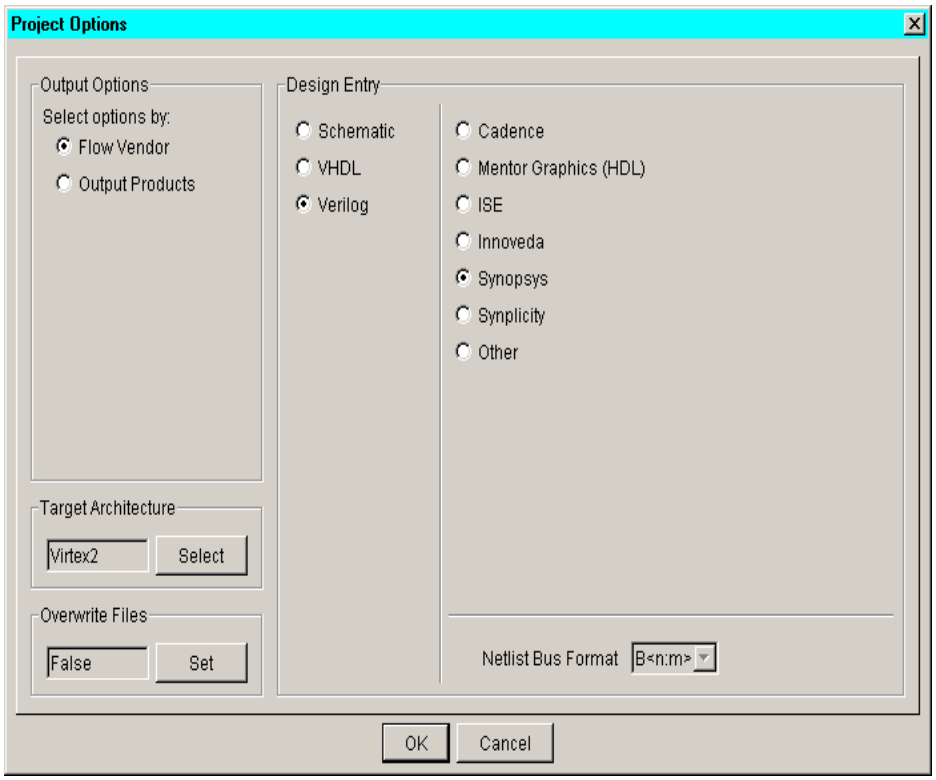


Figure 5-3: **Sample Project Options Setting for Synopsys Verilog Flow**

In the preceding figure, the Design Entry is set to **Verilog**. If the Vendor is set to **Synopsys**, the bus delimiter format is automatically set to B<n:m>. Table 5-3 shows the bus delimiter format associated with each synthesis vendor.

Table 5-3: **Bus Delimiter Format**

Vendor	EDIF Bus Delimiter Format
Mentor Graphics (LeonardoSpectrum)	B(n:m)
Synopsys FPGA Compiler II	B<n:m>
Synplicity (Synplify)	B(n:m)
ISE (Xilinx XST)	B<n:m>
Innoveda (eProduct)	BI
Cadence	B<n:m>

3. Instantiate the module in the parent design.

   Insert the instantiation template from the VEO file into the parent design, and edit the module connections. 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 according to the instructions in the following section.

Verilog myadder8.veo Instantiation Template File

```
/*****************************************************************
* This file is owned and controlled by Xilinx and must be used     *
* solely for design, simulation, implementation and creation of    *
* design files limited to Xilinx devices or technologies. Use      *
* with non-Xilinx devices or technologies is expressly prohibited  *
* and immediately terminates your license.                         *
*                                                                  *
* Xilinx products are not intended for use in life support         *
* appliances, devices, or systems. Use in such applications are    *
* expressly prohibited.                                            *
*                                                                  *
* Copyright (C) 2001, Xilinx, Inc.  All Rights Reserved.           *
*****************************************************************/
// 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_IN(C_IN),
  .Q(Q),
  .CLK(CLK));

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

// You must compile the wrapper file myadder8.v when simulating
// the core, myadder8. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library.
// For detailed instructions, please refer to the
// "Coregen Users Guide".
```

The customized core myadder8 is defined in the wrapper file, myadder8.v, which is generated by Core Generator.

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

Verilog Parent Design File: myadder8_top.v

```
//-----------------------------------------------
module myadder8_top (A_P, B_P, C_INP, Q_P, CLK_P);
input [7 : 0] A_P;
input [7 : 0] B_P;
input C_INP;
output [8 : 0] Q_P;
input CLK_P;
// INST_TAG
myadder8 uut(
.A(A_P),
.B(B_P),
.C_IN(C_INP),
.Q(Q_P),
.CLK(CLK_P));
// INST_TAG_END
endmodule
```

4. Create a test bench.

   Create a test bench file called to simulate the parent design containing the myadder8 core. Include an instantiation of the parent design and stimuli to activate the adder. The following example displays the framework for a test bench used to simulate this design, with some sample simulation stimuli.

adder_tb.v File

```
‘timescale 1 ns/1 ps
module adder_tb;
reg CLKT;
reg C_INT;
reg [7:0] AT;
reg [7:0] BT;
wire [8:0] QT;
/* Instantiation of top level design */
myadder8_top uut (
    .A_P(AT),
    .B_P(BT),
    .C_INP(C_INT),
    .Q_P(QT),
    .CLK_P(CLKT)
);

/* Add stimulus here */
always #10 CLKT = ~CLKT;
initial begin
$timeformat(-9,3,"ns",12);
end

initial begin
C_INT = 0;
AT = 0;
BT = 0;
CLKT = 1;
#100
AT = 8’b10000000;
BT = 8’b00000001;
#40;
AT = 8’b11100001;
#40
BT = 8’b00000010;
#1000 $stop;
// #1000 $finish;
end
/* end stimulus section */
endmodule
```

5. Analyze the behavioral simulation.

   Analyze the simulation netlist, being sure to include the V wrapper file for the core. The following table describes compile and simulation commands for all Xilinx™ supported simulation vendors.

Table 5-4: **Simulation Instructions**

Vendor	Setup File Settings	Commands
MTI	None	vlib work vmap XilinxCoreLib <*compiled_XilinxCoreLib*> vlog adder_tb.v myadder8_top.v myadder8.v vsim -Lf XilinxCoreLib adder_tb
NC-Verilog	In cds.lib, add DEFINE XilinxCoreLib [compiled_XilinxCoreLib]	ncvlog adder_tb.v myadder8_top.v myadder8.v ncelab -message adder_tb ncsim -run adder_tb
VCS / VCSi	None	vcs -Mupdate -Mlib=<*compiled_XilinxCoreLib*> -y $XILINX / verilog / src / XilinxCoreLib +libext+.v +incdir+ $XILINX / verilog / src -R adder_tb.v myadder8_top.v myadder8.v
Verilog-XL	None	verilog +incdir+$XILINX / verilog / src -y <*src_XilinxCoreLib*> +libext+.v adder_tb.v myadder8_top.v myadder8.v

6. Synthesize the design.

   Most synthesis tools automatically infer a black box upon encountering a Verilog™ module declaration, which only contains port directional declarations. The logic for each core is specified in its EDIF implementation netlist (*component_name*.EDN) and, for some cores, may also be specified in additional NGC files, but it is not specified in any Verilog file.

   Note: For Synplicity™ only, direct the synthesizer to treat each core as a black box if necessary.

   Synthesize the parent design following the vendor specific instructions in the following table.

Table 5-5: **Synthesis Tool Instructions**

Vendor Tool	Instructions
Mentor Graphics LeonardoSpectrum	Do not read in a separate V or EDIF file for the CORE Generator module. Mentor Graphics automatically treats the module as a black box.
Synopsys FPGA Compiler II	No special instructions.

Table 5-5: **Synthesis Tool Instructions**

Vendor Tool	Instructions
Synplicity™ Synplify™	Analyze the V wrapper file in Synplify. The CORE Generator™ attaches the appropriate black_box attribute to the core's module declaration in the V wrapper file for the module. Analyzing the V wrapper file is sufficient to prevent black box warnings from Synplify during compilation.
ISE (Xilinx XST)	No special instructions.

The following example shows how the CORE Generator configures the module declaration for the core as a Verilog black box for Synplicity.

Synplicity Verilog Black Box

```
module myadder8 (
  A,
  B,
  C_IN,
  Q,
  CLK);    // synthesis black_box

input [7 : 0] A;
input [7 : 0] B;
input C_IN;
output [8 : 0] Q;
input CLK;

// (list of customization parameters omitted)

  inst (
    .A(A),
    .B(B),
    .C_IN(C_IN),
    .Q(Q),
    .CLK(CLK));

// (other optional 3rd party black box attributes
// omitted)

endmodule
```

7. Write out the Implementation Netlist.

   After the parent design has been synthesized, write out its implementation netlist using the synthesis tool. The implementation netlist file is written out in EDIF (.EDF) format.

   You have the option of either breaking buses out into their individual bus bits(B<I>, B(I), B[I], or BI), or maintaining them as a single array (B<n:m>, B(n:m) or B[n:m]) when writing out the EDIF implementation netlist for a module.

   Before generating cores for an HDL design, you should confirm that you have set the bus format (individual bus bits or a single array) to match the format you use for the ports when you instantiate the core.

For lists of the various vendor tools and descriptions for writing out netlists, see the following table:

Table 5-6: **Implementation Netlist Formats**

Vendor	Description
Mentor Graphics™ LeonardoSpectrum™	Writes out EDIF netlist by default.
Synopsys™ FPGA Compiler II™	FPGA Compiler II writes out an EDIF file for all Xilinx designs.
Synplicity™ Synplify™	Synplify writes out EDIF for all Xilinx 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 Project Navigator) 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 (and NGC, if applicable) 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 place all the EDIF and NGC files associated with the design in a separate directory and then run NGDBuild with the **-sd** option to specify explicitly the location of the directory containing the CORE Generator implementation netlist files.

Creating VHDL Designs

This section describes the procedure for behavioral simulation, synthesis, and implementation of VHDL designs containing CORE Generator™ modules. To integrate a CORE Generator module into a VHDL design, refer to the design flow in the following figure.

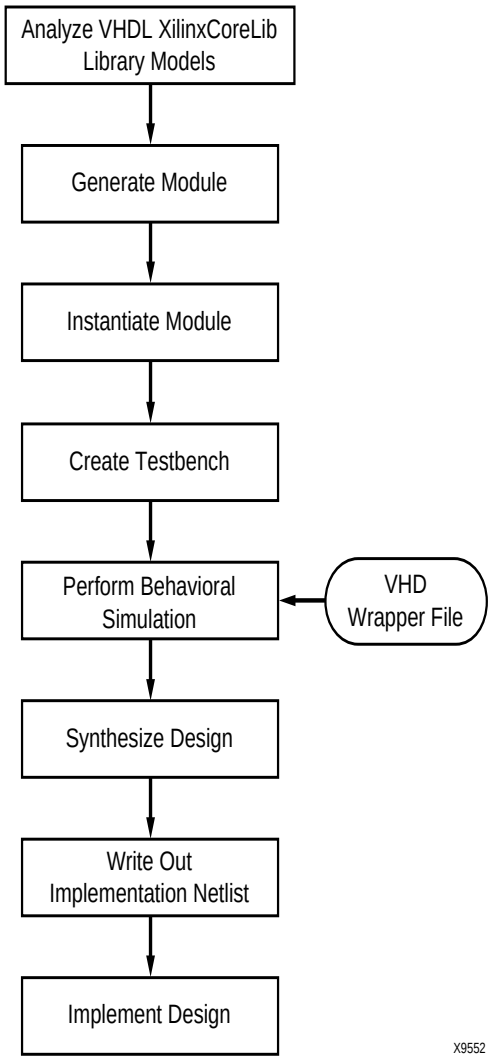


Figure 5-4: **VHDL Front End Design Flow Chart**

VHDL HDL Design Flow

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

Table 5-7: **Vendor Tools for VHDL Flow**

Function	Tool
Synthesis	Synopsys™ FPGA Compiler II™ Synplicity™ Synplify™ Mentor Graphics™ LeonardoSpectrum™ Xilinx™ XST
Simulation	MTI™ ModelSim / VCOM

VHDL Design Flow Procedure

This section describes the detailed procedure for the VHDL design flow.

1. Compiling VHDL Behavioral Simulation Library

   All VHDL simulators require that VHDL models be analyzed before simulation can actually proceed. The VHDL source models for Xilinx CORE Generator modules are located in $XILINX / vhdl / src / XilinxCoreLib.

   Support for the XilinxCoreLib library is available in the compxlib compilation script located in $XILINX / bin / <*platform*>. Use compxlib to compile the XilinxCoreLib VHDL library. For more information on compxlib, refer to "HDL Behavioral Simulation Flow Features".

   You can alternatively compile XilinxCoreLib by directly invoking the individual MTI commands as follows.

   a. Create the XilinxCoreLib library with MTI ModelSim / VHDL selected, by entering the following commands:

      ```
      cd library_directory
      vlib xilinxcorelib
      ```

      Note: The name of the analyzed library, xilinxcorelib, must be lowercase.

   b. Establish a link to the compiled behavioral models. To use a library of compiled behavioral models in your simulation, a link must be established between the compiled source in your project directory and the compiled library directory. To map the XilinxCoreLib library to your project directory, type the following:

      ```
      vmap xilinxcorelib library_directory/xilinxcorelib
      ```

      This maps the logical name of xilinxcorelib to the XilinxCoreLib library directory created by the *vlib* command.

      The vmap command creates and also modifies the MTI modelsim.ini file. This file is read by the ModelSim / VHDL simulator, which uses the library mapping information in it to map library names to physical locations on a disk or network.

   c. Analyzing the Behavioral Models

Analyze the VHDL models into the previously mapped Xilinxcorelib library in the order specified in the vhdl_analyze_order file. The following excerpt is an example from the vhdl_analyze_order file:

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

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

ul_utils.vhd
vfftv2_utils.vhd
vfft1024v2.vhd
vfft1024v2_comp.vhd
vfft16v2.vhd
vfft16v2_comp.vhd
vfft256v2.vhd
.
.
.
mulVHT.vhd
multVHT_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 proper order, specifically, primitive models before macro level models. Compiling the models in the wrong order leads to errors in compilation.

2. Generating the Module

   To set up a new CORE Generator project, follow the procedure in "Creating a New Project" in Chapter 3. CORE Generator cores can only be created in a CORE Generator project.

a. Select **Project** → **Project Options**. Click **Flow Vendor** under **Output Options**. The Design Entry panel appears in the right hand side of the dialog.

b. From the Design Entry panel, select **VHDL** in the first column for the Design Flow, and your target Vendor in the second column. This specifies the vendor software you will use for your VHDL design, and the preferred EDIF bus delimiter format for compatibility with the synthesis tool output. Setting these options correctly ensures proper integration with the upper level parent implementation netlist.

The vendor setting changes the bus delimiter displayed in the Netlist Bus Format box to the correct setting for that vendor. See Table 5-3 for information on the specific bus delimiter format for each synthesis vendor.

As shown in the following figure, if the Vendor is set to ISE, the bus delimiter format is automatically set to B<n:m>.

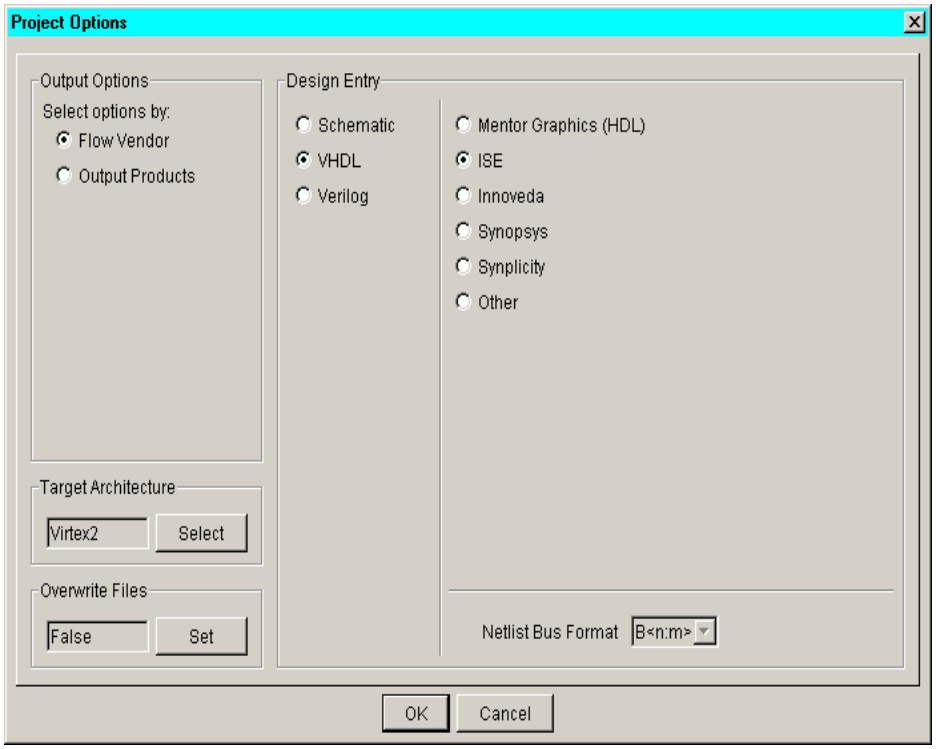


Figure 5-5: **VHDL Behavioral Simulation Option**

3. Instantiating the Module

When you select **VHDL** as your design flow, a VHDL template file *component_name*.VHO, an implementation netlist *component_name*.EDN, and (for some cores) NGC netlists, are generated. This occurs 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

For more information on the VHO file, see "Using a VHO Instantiation Template File".

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

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

b. Modify the port connections in the instantiation template to reflect the actual connections to the parent design. For more details, see "VHDL Parent Design File myadder8_top.vhd".

   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 core is integrated properly when the parent VHDL design has been synthesized. The VHDL instantiation of the core in the parent design serves as a placeholder for the core. After the parent design has been synthesized, the Xilinx tools merge the core's EDIF netlist (and, for some cores, any underlying NGC files) 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.

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, the myadder.vhd wrapper file, and the parent design, myadder8_top.vhd.

VHDL Template File myadder8.vho

```
------------------------------------------------------------------------
-- This file is owned and controlled by Xilinx and must be used     --
-- solely for design, simulation, implementation and creation of    --
-- design files limited to Xilinx devices or technologies. Use      --
-- with non-Xilinx devices or technologies is expressly prohibited  --
-- and immediately terminates your license.                         --
--                                                                  --
-- Xilinx products are not intended for use in life support         --
-- appliances, devices, or systems. Use in such applications are    --
-- expressly prohibited.                                            --
--                                                                  --
-- Copyright (C) 2001, Xilinx, Inc.  All Rights Reserved.           --
------------------------------------------------------------------------

-- 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: IN std_logic;
  Q: OUT std_logic_VECTOR(8 downto 0);
  CLK: IN std_logic);
end component;

-- XST black box declaration
attribute box_type : string;
attribute box_type of myadder8: component is "black_box";

-- FPGA Compiler Black Box declaration
attribute fpga_dont_touch: string;
attribute fpga_dont_touch of myadder8: component is "true";

-- Synplicity black box declaration
```

```
attribute black_box : boolean;
attribute black_box of myadder8: component is true;

-- 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_IN => C_IN,
     Q => Q,
     CLK => CLK);
-- INST_TAG_END ------ End INSTANTIATION Template ------------

-- You must compile the wrapper file myadder8.vhd when simulating
-- the core, myadder8. When compiling the wrapper file, be sure to
-- reference the XilinxCoreLib VHDL simulation library. For detailed
-- instructions, please refer to the "Coregen Users Guide".
```

The wrapper file myadder8.vhd is generated by the CORE Generator™ along with an EDIF netlist and VHO file when VHDL outputs are requested. The wrapper file contains a Configuration Specification which binds the customized core to the corresponding XilinxCoreLib VHDL behavioral simulation model, and also passes the VHDL model generics which customize the model. The wrapper file must be analyzed during behavioral simulation.

Note: In this example of the Adder core, no NGC netlist files are produced by the core. Some other cores produce NGC files in addition to the EDIF netlist.

Since the 4.1i release, with the introduction of the Configuration Specification construct, you no longer need to use configuration declarations in your upper level VHDL blocks.

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

VHDL Parent Design File myadder8_top.vhd

```
library IEEE;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_1164.all;

ENTITY myadder8_top IS

PORT (
    AP:  IN std_logic_vector(7 downto 0);
    BP: IN std_logic_vector(7 downto 0);
    C_INP: IN std_logic ;
    QP: OUT std_logic_VECTOR (8 downto 0);
    CLKP: IN  std_logic);

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: IN std_logic;
     Q: OUT std_logic_VECTOR(8 downto 0);
     CLK: IN std_logic
     );
end component;

-- Synplicity black box declaration
attribute black_box : boolean;
attribute black_box of myadder8: component is true;

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_1".
------------------------------------------------------------
myadder8_1     : myadder8

port map (
        A => AP,
        B => BP,
        C_IN => C_INP,
        Q => QP,
        CLK => CLKP
        );
end use_core;
```

4. Creating the Test Bench

   Create a test bench file to simulate a parent design containing the myadder8 core. The test bench must instantiate the parent design. The test bench should also contain stimulus to activate the adder. The following example displays a part of the test bench file used to simulate this design. In this example, the section containing simulation stimulus is omitted.

VHDL Test Bench File myadder_tb.vhd

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

ARCHITECTURE simulate OF myadder_tb 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);
    CLKP: IN std_logic ;
    C_INP: IN std_logic;
    QP: 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 carry_in : std_logic;
SIGNAL sum : std_logic_vector (8 DOWNTO 0);

BEGIN
uut: myadder8_top
          PORT MAP (
     AP => a_data_input,
     BP => b_data_input,
     CLKP => clock,
     C_INP=> carry_in,
     QP => Q);
stimulus: PROCESS
          BEGIN

-------------------------------------------------------
----Provide stimulus in this section. (not shown here)
-------------------------------------------------------
          wait;
          end process; -- stimulus
END simulate;
```

5. Performing Behavioral Simulation

   Before the Model Technology™ simulation tools can be used to simulate the design, the wrapper file for the module, the parent design, and the test bench 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 wrapper file, the parent design and test bench file. Start up MTI ModelSim™ in the *project_directory* and type the following:

      ```
      vlib work
      vcom myadder8.vhd
      vcom myadder8_top.vhd
      vcom myadder_tb.vhd
      ```

b. Invoke the simulator.

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

```
vsim myadder_tb
```

6. Synthesizing the Design

Synthesize the parent design containing the core or cores. Follow the vendor specific instructions in Table 5-8 for integrating a black box module into your design. If needed, please refer to your synthesis vendor documentation for more details.

In most cases *you must NOT add the VHD wrapper for the core to your synthesis project* or analyze it as part of your synthesis flow. Additionally, although the CORE Generator™ VHO template files continue to specify black box attributes for the supported synthesis vendors, attachment of black box attributes is now optional for most vendors.

Note: Implementation logic for a CORE Generator core is specified in its EDIF implementation netlist <*component_name*>.EDN and, for some cores, may also be specified in additional NGC files, but it is not specified in the VHD wrapper file for the core.

The following table gives special instructions for the different synthesis tools:

Table 5-8: **Synthesis Tool Descriptions**

Vendor Tool	Special Instructions
Mentor Graphics™ LeonardoSpectrum™	Do not read in a separate VHD or EDIF file for the CORE Generator module. Mentor Graphics automatically treats the module as a black box.
Synopsys™ FPGA Compiler II™	No special instructions.
Synplicity™ Synplify™	Do not read in a separate VHD or EDIF file for the CORE Generator module. It is also recommended that you attach a syn_black_box attribute to the component declaration for the CORE Generator module as indicated in the VHO template generated for the core. This attribute is optional but prevents Synplicity from issuing warnings about black box modules.
ISE (Xilinx XST)	No special instructions.

VHDL Black Box

```
component myadder8
     port (
     A: IN std_logic_VECTOR(7 downto 0);
     B: IN std_logic_VECTOR(7 downto 0);
     C_IN: IN std_logic;
     Q: OUT std_logic_VECTOR(8 downto 0);
     CLK: IN std_logic
     );
end component;

-- Synplicity black box declaration
attribute black_box : boolean;
attribute black_box of myadder8: component is true;
```

7. Writing out the Implementation Netlist

   After the parent design has been synthesized, write out its implementation netlist using the synthesis tool.

   You have the option of either breaking buses out into their individual bus bits(B<I>, B(I), B[I], or BI), or maintaining them as a single array (B<n:m>, B(n:m) or B[n:m]) when writing out the EDIF implementation netlist for a module.

   Before generating cores for an HDL design, you should confirm that you have set the bus format (individual bus bits or a single array) to match the format you use for the ports when you instantiate the core.

   For lists of the various vendor tools and descriptions for writing out netlists, see the following table:

Table 5-9: **Implementation Netlist Formats**

Vendor	Description
Mentor Graphics™ LeonardoSpectrum™	Writes out EDIF netlist by default. No special instructions.
Synopsys™ FPGA Compiler II™	FPGA Compiler II writes out an EDIF file for all Xilinx™ designs. No special instructions.
Synplicity Synplify	Synplify writes out EDIF for all Xilinx designs.

8. 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 runs on the top level parent design during design implementation. To merge the netlists successfully, verify that all of the CORE Generator EDN EDIF (and NGC, if applicable) 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 place all the EDIF and NGC files associated with the design in a separate directory and run NGDBuild with the **-sd** option to specify explicitly the location of the directory containing the CORE Generator implementation netlist files.

Using Instantiation Templates

Instantiation template files are files containing code that can be used to instantiate a CORE Generator module into your Verilog™ or VHDL design. Verilog instantiation template files have an extension of .VEO, and VHDL instantiation template files have an extension of .VHO.

Using a VEO Instantiation Template File

Starting with the 4.1i release, the VEO file has been simplified so that it only contains an instantiation template. The module declaration and model customization parameters that previously were included in the VEO file are now delivered in a separate V wrapper file.

The following is an example of the VEO file for a customized 8 bit adder module called adder8:

Verilog Instantiation Template for an 8-Bit Adder

```
// 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
adder8 YourInstanceName
    .A(A),
    .B(B),
    .C(C),
    .CE(CE),
    .CI(CI),
    .CLR(CLR),
    .S(S));
// INST_TAG_END ------ End INSTANTIATION Template ---------

// You must compile the wrapper file test.v when simulating
// the core, test. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library.
// For detailed instructions, please refer to the
// "Coregen Users Guide".
```

Verilog Wrapper file for adder8: adder8.v

```
// The synopsys directives "synopsys translate_off/translate_on"
// specified below are supported by XST, FPGA Compiler, Exemplar
// and Synplicity synthesis tools. Ensure they are correct
// for your synthesis tool(s).

// You must compile the wrapper file adder8.v when simulating
// the core, adder8. When compiling the wrapper file, be sure to
// reference the XilinxCoreLib Verilog simulation library.
// For detailed instructions, please refer to the
// "Coregen Users Guide".

module adder8 (
  A,
  B,
  C_IN,
  Q,
  CLK);    // synthesis black_box

input [7 : 0] A;
input [7 : 0] B;
input C_IN;
output [7 : 0] Q;
input CLK;

// synopsys translate_off

  C_ADDSUB_V4_0 #(
    0,// c_ainit_val
    "0000",// c_a_type
    1,// c_a_width
    8,// c_bypass_enable
    0,// c_bypass_low
    0,// c_b_constant
```

```
    0,// c_b_type
    1,// c_b_value
    "0",// c_b_width
    8,// c_enable_rlocs
    1,// c_has_aclr
    0,// c_has_add
    0,// c_has_ainit
    0,// c_has_aset
    0,// c_has_a_signed
    0,// c_has_bypass
    0,// c_has_b_in
    0,// c_has_b_out
    0,// c_has_b_signed
    0,// c_has_ce
    0,// c_has_c_in
    1,// c_has_c_out
    0,// c_has_ovfl
    0,// c_has_q
    1,// c_has_q_b_out
    0,// c_has_q_c_out
    0,// c_has_q_ovfl
    0,// c_has_s
    0,// c_has_sclr
    0,// c_has_sinit
    0,// c_has_sset
    0,// c_high_bit
    7,// c_latency
    1,// c_low_bit
    0,// c_out_width
    8,// c_pipe_stages
    1,// c_sinit_val
    "0",// c_sync_enable
    0,// c_sync_priority
    1)// c_sync_priority
  inst (
    .A(A),
    .B(B),
    .C_IN(C_IN),
    .Q(Q),
    .CLK(CLK));


// synopsys translate_on

// FPGA Compiler black box declaration
// synopsys attribute fpga_dont_touch "true"
// synthesis attribute fpga_dont_touch of adder8 is "true"

// XST black box declaration
// box_type "black_box"
// synthesis attribute box_type of adder8 is "black_box"

endmodule
```

Using a VHO Instantiation Template File

A VHO file contains code that you can use to instantiate a CORE Generator™ module in your VHDL design. The VHDL configuration declaration, which used to be in the VHO file in previous releases, has been replaced by a *configuration specification*. The VHDL configuration specification is declared in the wrapper file, module_name.vhd, for the core. This arrangement hides the code complexity associated with the customization of the parameterized simulation model for the core. You can process the wrapper file the same way you would process a simulation model, without worrying about the internal contents of the file. Like an actual simulation model, the VHD wrapper file must be analyzed during simulation.

With this enhancement, it is no longer necessary to add a configuration declaration in your upper level designs.

The following is an example of the VHDL instantiation template and wrapper file for a customized 8 bit adder, adder8.

VHDL Instantiation Template for adder8. (adder8.vho)

```
-- The following code must appear in the VHDL architecture header:

------------- Begin Cut here for COMPONENT Declaration ------ COMP_TAG

component adder8
  port (
  A: IN std_logic_VECTOR(7 downto 0);
  B: IN std_logic_VECTOR(7 downto 0);
  C_IN: IN std_logic;
  Q: OUT std_logic_VECTOR(7 downto 0);
  CLK: IN std_logic);
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 : adder8
    port map (
      A => A,
      B => B,
      C_IN => C_IN,
      Q => Q,
      CLK => CLK);
-- INST_TAG_END ------ End INSTANTIATION Template ------------

-- You must compile the wrapper file adder8.vhd when simulating
-- the core, adder8. When compiling the wrapper file, be sure to
-- reference the XilinxCoreLib VHDL simulation library. For detailed
-- instructions, please refer to the "Coregen Users Guide".
```

VHDL Wrapper File for adder8: adder8.vhd

```
-- You must compile the wrapper file adder8.vhd when simulating
-- the core, adder8. When compiling the wrapper file, be sure to
-- reference the XilinxCoreLib VHDL simulation library. For detailed
-- instructions, please refer to the "Coregen Users Guide".

-- The synopsys directives "synopsys translate_off/translate_on"
-- specified below are supported by XST, Exemplar and Synplicity
-- synthesis tools. Ensure they are correct for your
-- synthesis tool(s).

-- synopsys translate_off
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

Library XilinxCoreLib;

ENTITY adder8 IS
  port (
  A: IN std_logic_VECTOR(7 downto 0);
  B: IN std_logic_VECTOR(7 downto 0);
  C_IN: IN std_logic;
  Q: OUT std_logic_VECTOR(7 downto 0);
  CLK: IN std_logic);
END adder8;

ARCHITECTURE adder8_a OF adder8 IS

component wrapped_adder8
  port (
  A: IN std_logic_VECTOR(7 downto 0);
  B: IN std_logic_VECTOR(7 downto 0);
  C_IN: IN std_logic;
  Q: OUT std_logic_VECTOR(7 downto 0);
  CLK: IN std_logic);
end component;

-- Configuration specification
  for all : wrapped_adder8 use entity
  XilinxCoreLib.C_ADDSUB_V4_0(behavioral)
    generic map(
      c_has_ainit => 0,
      c_has_s => 0,
      c_sync_enable => 0,
      c_has_q => 1,
      c_has_sinit => 0,
      c_has_sset => 0,
      c_has_add => 0,
      c_has_ovfl => 0,
      c_has_q_b_out => 0,
      c_has_sclr => 0,
      c_out_width => 8,
      c_sinit_val => "0",
      c_bypass_low => 0,
      c_has_b_signed => 0,
      c_b_constant => 0,
      c_has_bypass => 0,
```

```
      c_low_bit => 0,
      c_a_type => 1,
      c_has_aset => 0,
      c_has_q_c_out => 0,
      c_b_type => 1,
      c_add_mode => 0,
      c_has_q_ovfl => 0,
      c_has_aclr => 0,
      c_has_b_in => 0,
      c_has_c_in => 1,
      c_has_b_out => 0,
      c_latency => 1,
      c_pipe_stages => 1,
      c_sync_priority => 1,
      c_b_width => 8,
      c_b_value => "0",
      c_bypass_enable => 0,
      c_has_a_signed => 0,
      c_has_c_out => 0,
      c_enable_rlocs => 1,
      c_a_width => 8,
      c_has_ce => 0,
      c_high_bit => 7,
      c_ainit_val => "0000");

BEGIN

U0 : wrapped_adder8
    port map (
      A => A,
      B => B,
      C_IN => C_IN,
      Q => Q,
      CLK => CLK);
-- synopsys translate_on

END adder8_a;
```

Chapter 6

The Memory Editor

This chapter describes the Memory Editor. The chapter includes the following sections:

Memory Editor Overview

The Memory Editor is a tool that helps to create COE files to specify memory contents and initialization values for CORE Generator™ memory cores. Although other COE files may be used for other purposes (specifying FIR filter coefficients, for example), the Memory Editor generates COE files formatted for CORE Generator memory cores only.

A single memory is typically made up of one or more memory blocks. For each memory, the Memory Editor creates a single CGF file which defines the contents of one or more COE files. For each memory block defined in a CGF file you define in the Memory Editor, the Memory Editor generates a separate COE file.

The Memory Editor COE Generation Format (CGF) file is a dual purpose log and specification file. As a log file, it records the user-specified inputs that are used to generate the COE files for the memory. As a specification file, it can be used to define the contents of COE files for memory blocks. A pre-existing CGF file can be edited, saved, and then loaded into the Memory Editor and used to create a new COE file or files.

The Memory Editor is accessed by selecting **Tools** → **Memory Editor** in the CORE Generator GUI.

The Memory Editor GUI

When the Memory Editor is invoked from the CORE Generator™ GUI, a Memory Editor Control Panel and a Memory Contents window open (shown following).

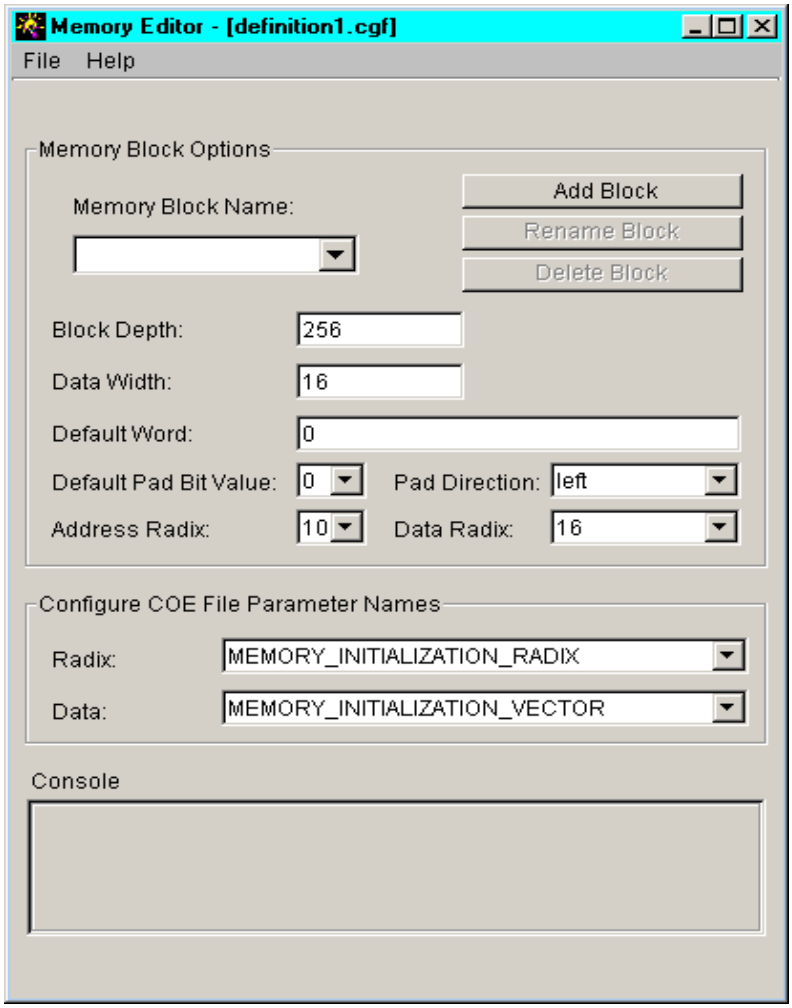


Figure 6-1: **Memory Editor Control Panel**

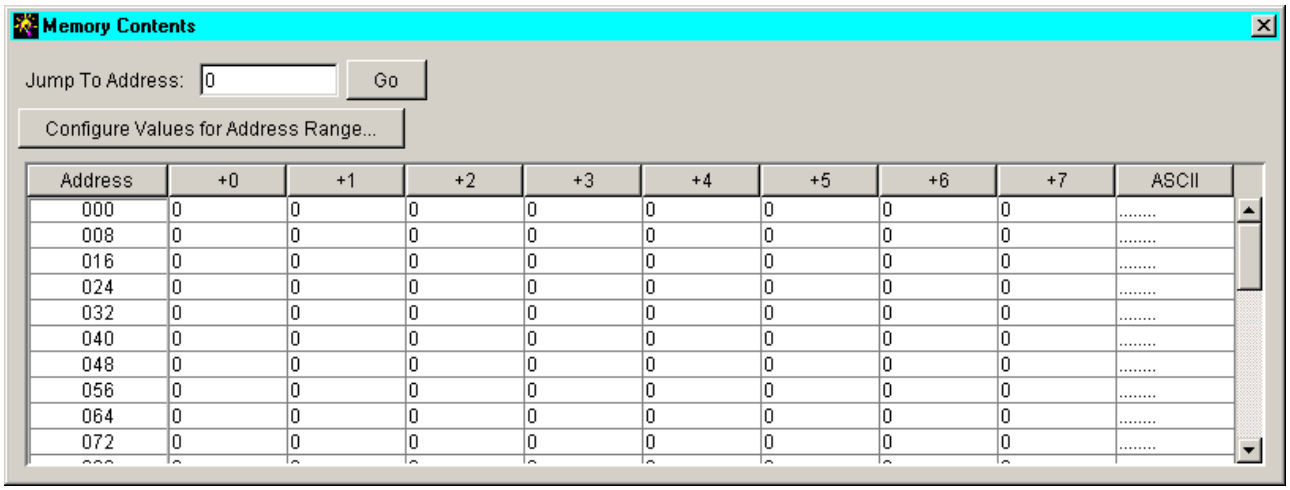


Figure 6-2: **Memory Contents Window**

The fields in the Memory Editor Control Panel (Figure 6-1) are described in the following tables. The “CGF Parameter” column shows the corresponding CGF file value.

Table 6-1: **Individual Memory Block Properties**

GUI Field (Memory Editor Control Panel)	Default Value	CGF Parameter	Description
Memory Block Name	none	#memory_block_name	Name of the memory block. A memory block name cannot start with a digit and can only contain the characters 'a-z', '0-9', and '_'.
Block Depth	256	#block_depth	Memory depth. The number of words in the memory block. The maximum depth allowable is 1048576.
Data Width	16	#data_width	The number of bits making up a memory word.
Default Word	0	#default_word	Default value of a memory word if no explicit value is specified.

Table 6-1: **Individual Memory Block Properties**

GUI Field (Memory Editor Control Panel)	Default Value	CGF Parameter	Description
Default Pad Bit Value	0	#default_pad_bit_value	Used as a default value to pad the memory values when the width of the data value you provide is less than the Data Width of the memory. Options: 0, 1
Pad Direction	left	#pad_direction	The direction in which a Data Word is padded (using the default bit value) when the width of the data word you provide is less than the Data Width of the memory. Options: left, right
Data Radix	16	#data_radix and #signed	Radix used to specify the value of data words (for example, binary or hexadecimal). When Data Radix is set to a value of 10 (signed) or 10 (unsigned) in Memory Editor Control Panel, #signed CGF parameter is set to true or false. Options: 2, 8, 10 (signed), 10 (unsigned), 16
Address Radix	10	#address_radix	The radix used for specifying memory address values (for example, binary or hexadecimal). Options: 2, 8, 10, 16

Table 6-1: **Individual Memory Block Properties**

GUI Field (Memory Editor Control Panel)	Default Value	CGF Parameter	Description
Radix (under Configure COE File Parameter Names)	MEMORY_INITIALIZATION_RADIX	#coe_radix	Keyword that will be written out to COE file for memory word radix. The data radix specifies memory initialization values. To determine the memory word radix keyword required for a CORE Generator core, refer to the data sheet for that core.
Data (under Configure COE File Parameter Names)	MEMORY_INITIALIZATION_VECTOR	#coe_data	Keyword that will be written out to COE file for memory data values. The data vector represents the actual array or vector of memory values. To determine the memory data value keyword required for a CORE Generator core, refer to the data sheet for that core.

Table 6-2: **Memory Block Operations**

GUI Field	Description
Add Block	To add a new memory block.
Rename Block	To rename a memory block.
Delete Block	To delete a memory block. This has an associated confirm dialog to prevent accidentally losing a large block of data in one keystroke.

The fields in the Memory Contents Window (Figure 6-2) are described in the following table.

Table 6-3: **Memory Contents Window Fields**

GUI Field (Contents Window)	Description
Jump to Address	Allows you to display a memory location in the Address table. When you enter an address in this box and click Go, the address table adjusts to display the value at the address.
Go	Adjusts address table to display the address in the Jump to Address field.
ConfigureValues for Address Range	Allows you to set a memory value for a range of addresses. When you click this button, a dialog box appears for you to specify the start address, the end address, and the value you want to set for the range of addresses.
Address table	Shows the values assigned to the different memory addresses.
ASCII	Shows the ASCII equivalents for the memory values in each cell within an address table row. A period indicates that the data in the memory cell does not specify a valid ASCII character. If the memory data has a Data Width of 8, you can enter ASCII characters in the ASCII column and the equivalent values will automatically be written in the Address table.

Creating a Memory with a Single Memory Block

To create a memory with a single memory block:

1. In the Core Generator™ window, select **Tools** → **Memory Editor**.
2. In the Memory Editor Control Panel, select **File** → **New Memory Definition**.
3. In the Memory Editor Control Panel, click the **Add Block** button.
4. In the Add Block dialog box, enter a memory block name and click **OK**.
5. Configure the fields of the Memory Editor Control Panel.

   These fields are described in Table 6-4.

Table 6-4: **Memory Editor Control Panel Fields**

Field	Description
Block Depth	The number of data words in the memory block. The maximum depth allowable is 1048576.
Data Width	The number of bits making up a memory word.
Address Radix	The radix used for specifying memory address values (for example, binary, hexadecimal).
Default Word	Default value of a memory word if no explicit value is specified.

Table 6-4: **Memory Editor Control Panel Fields**

Field	Description
Default Pad Bit Value	Used as a default value to pad the memory values when the width of the data value you provide is less than the Data Width of the memory.
Pad Direction	The direction in which a Data Word is padded (using the Default Pad Bit Value) when the width of the data word you provide is less than the Data Width of the memory.
Address Radix	The radix used for specifying memory address values (for example, binary or hexadecimal).
Data Radix	The radix used to specify the value of data words (for example, binary or hexadecimal). If you are specifying a Data Radix of 10 (decimal), you can select 10 (signed) or 10 (unsigned). If the values are signed, the MSB is the sign bit.
Radix (under Configure COE File Parameter Names)	Keyword that will be written out to COE file for memory word radix. The data radix specifies memory initialization values. To determine the memory word radix keyword required for a CORE Generator core refer to the data sheet for that core.
Data (under Configure COE File Parameter Names)	Keyword that will be written out to COE file for memory data values. The data vector represents the actual array or vector of memory values. To determine the memory data value keyword required for a CORE Generator core refer to the data sheet for that core.

6. In the Memory Contents window, set the values for the memory locations in this way:
   - To set the value for a single location, click the cell for the location and enter the value.
   - To set the value for a location not currently displayed, enter the address in the **Jump to Address** box and click **Go**. The location is displayed in the address table below. You can then set the value for that location.
   - To set the values for a range of addresses, click the **Configure Values for Address Range** button, enter the Start Address, End Address, and Value in the dialog box that opens, and click **OK**.

   Values must be set to conform to the options (Data Width, Pad Bit Value, and others) set in the Memory Editor Control Panel.
7. In the Memory Editor Control Panel, select **File** → **Generate**.
8. In the Generate dialog box, select **COE File(s) (for CORE Generator)** and enter a directory in which the COE file will be saved.
9. If you want to save the data in a CSV (Comma delimited) file, select **CSV File(s)**.
10. Click **OK**.

    The Memory editor writes a CGF file with the specified name and a COE file with the name *cgffilename_blockname*.coe, where *cgffilename* is the name of the CGF file and *blockname* is the Memory Block Name specified in the Memory Editor Control Panel.

"Sample CGF and COE Files – Single Memory Block" shows the format of a CGF file and a COE file for a memory with a single memory block.

Adding Additional Memory Blocks to a Memory

You can build a memory one block at a time using the Add Block command in the Memory Options panel.

To add memory blocks to an existing memory:

1. In the Core Generator window, select **Tools** → **Memory Editor**.
2. In the Memory Editor Control Panel, select **File** → **Open Memory Definition**.
3. In the Open dialog box, open the CGF file to which you will add a memory block.
4. In the Memory Editor Control Panel, click the **Add Block** button.
5. In the Add Block dialog box, enter a memory block name and click **OK**.
6. Configure the fields in the Memory Editor Control Panel.

   These fields are described in Table 6-5.
7. In the Memory Contents window, set the values for the memory locations in this way:
   - To set the value for a single location, click the cell for the location and enter the value.
   - To set the value for a location not currently displayed, enter the address in the **Jump to Address** box and click **Go**. The location is displayed in the address table below. You can then set the value for that location.
   - To set the values for a range of addresses, click the **Configure Values for Address Range** button, enter the Start Address, End Address, and Value in the dialog box that opens, and click **OK**.

   Values must be set to conform to the options (Data Width, Pad Bit Value, and others) set in the Memory Editor Control Panel.
8. Continue to add blocks to the memory by performing steps 4 through 7 for each additional block.
9. When you have configured all of the additional blocks, select **File** → **Generate** in the Memory Editor Control Panel.
10. In the Generate dialog box, select **COE File(s) (for CORE Generator)** and enter a directory in which the COE files will be saved.
11. If you want to save the data in CSV (Comma delimited) files, select **CSV File(s)**.
12. Click **OK.**

    The Memory Editor saves the information about the new blocks in the CGF file and writes a COE file for each additional block. Each COE file has a name in the form *cgffilename_blockname*.coe, where *cgffilename* is the name of the CGF file and *blockname* is the Memory Block Name specified in the Memory Editor Control Panel.

"Sample CGF and COE Files – Multiple Memory Blocks" shows the format of a CGF file and the COE files for a memory with multiple memory blocks.

Specifying COE File Keywords

The Memory Editor defaults to the following two keywords for memory word radix and memory data values, respectively, when writing out COE files:

- MEMORY_INITIALIZATION_RADIX

  The data radix used to specify memory initialization values.

- MEMORY_INITIALIZATION_VECTOR

  The data vector representing the actual array or vector of memory values.

Other COE file keywords may be required for older versions of the Xilinx™ CORE Generator™ memory cores. To determine the specific keywords required for a CORE Generator core refer to the data sheet for that core. You can also specify COE file keywords other than the pre-programmed values by changing the Radix and Data fields in the Configure COE File Parameter Names area of the Memory Editor Control Panel.

Importing a CSV File

The Memory Editor can import a CSV (Comma delimited) file saved from Microsoft Excel and generate a COE file containing the memory data values specified in the CSV file.

The CSV import allows you to specify memory values in a Microsoft Excel spreadsheet, using mathematical operations and data concatenations in the cells of the spreadsheet instead of specifying each value individually.

To import a CSV file into the Memory Editor:

1. In the Memory Editor Control Panel, select **File** → **Import** → **CSV file**.
2. In the Open Memory Definition dialog box, type the name of the CSV file in the **File Name** text field

   OR

   Click the **Browse** button and navigate to the CSV file.
3. In the Open Memory Definition dialog box, click Open.
4. In the Memory Editor Settings dialog box, specify the Memory Depth, Word Width, Address Radix, Data Radix, and Start Address of the memory block.

   These fields are described in Table 6-5.

Table 6-5: **Memory Editor Settings Dialog Box Fields**

Field	Description
Memory Depth	The number of data words the CSV file represents. The maximum depth allowable is 1048576.
Word Width	The number of bits making up a memory word.
Address Radix	The radix used for specifying memory address values (for example, binary, hexadecimal).

Table 6-5: **Memory Editor Settings Dialog Box Fields**

Field	Description
Data Radix	The radix used to specify the value of data words (for example, binary or hexadecimal). If you are specifying a Data Radix of 10 (decimal), you can select 10 (signed) or 10 (unsigned). If the values are signed, the MSB is the sign bit.
Start Address	The starting address for the data words in the CSV file. Addresses in the Memory Editor will be filled sequentially starting at this address.

5. Click OK.

   In the Memory Editor Control Panel, the name of the CSV file appears as the Memory Block Name, and the settings specified when you imported the file appear in the Block Depth, Data Width, Address Radix, and Data Radix fields.

   In the Memory Contents Window, the memory locations will be filled in with the data from the CSV file. Locations will be filled in sequentially beginning with the Start Address you specified.

6. In the Memory Editor Control Panel, configure the other fields as needed: Default Word, Default Pad Bit Value, and Pad Direction.
7. Change the COE File Radix and Data keywords to the required values, if needed.
8. In the Memory Contents window, add or modify values as desired.

   Values must be set to conform to the options (Data Width, Pad Bit Value, and others) set in the Memory Editor Control Panel.

9. Select **File** → **Generate** in the Memory Editor Control Panel.
10. In the Generate dialog box, select **COE File(s) (for CORE Generator)** and enter a directory in which the COE file will be saved.
11. Click **OK.**

    The Memory Editor saves the information about the new block in the CGF file and writes a COE file for the block. The COE file has a name in the form *cgffilename_blockname*.coe, where *cgffilename* is the name of the CGF file and *blockname* is the Memory Block Name specified in the Memory Editor Control Panel.

Generating a CSV File

The Memory Editor can generate a CSV (Comma delimited) file for each memory block in the CGF file currently loaded in the Memory Editor Control Panel. The output CSV files can be opened in Microsoft™ Excel.

In an output CSV file, each Memory Editor data word will appear on a separate line. When this file is opened in Excel, each data word will appear in a single cell, and the cells will be arranged vertically one cell per row.

To generate CSV files representing the memory blocks in the Memory Editor:

1. In the Memory Editor Control Panel, select **File** → **Generate**.
2. In the Generate dialog box, select **CSV File(s)** and enter a directory in which the CSV files will be saved.
3. Click **OK.**

   The Memory Editor saves a CSV file for each memory block contained in the currently loaded CGF file. Each CSV file has a name in the form *cgffilename_blockname*.csv, where *cgffilename* is the name of the CGF file and *blockname* is the Memory Block Name specified in the Memory Editor Control Panel.

CGF File Format

The following is the general format of a CGF file:

Keyword=Value

Any lines following the string, **#data**, that do not start with a # sign are treated as data lines. Any data lines that start with the **@** sign are treated as address values.

Sample CGF and COE Files

The following sections show sample CGF files and COE files for a memory consisting of one block and a memory consisting of multiple blocks.

Sample CGF and COE Files – Single Memory Block

The examples below show the CGF file and COE file generated for a memory consisting of one block.

Sample CGF File Specifying a Single Memory Block (single.cgf)

```
#version2.0
#memory_block_name=tiger
#block_depth=14
#data_width=8
#default_word=0
#default_pad_bit_value=1
#pad_direction=left
#data_radix=16
#address_radix=10
#coe_radix=MEMORY_INITIALIZATION_RADIX
#coe_data=MEMORY_INITIALIZATION_VECTOR
#data=
@0
```

```
a
2b
3c
@11
28
12
3
#end
```

COE File Generated from single.cgf (single_tiger.coe)

```
MEMORY_INITIALIZATION_RADIX=2;
MEMORY_INITIALIZATION_VECTOR=
11111010,
00101011,
00111100,
11110000,
11110000,
11110000,
11110000,
11110000,
11110000,
11110000,
11110000,
11110000,
00101000,
00010010,
11110011;
```

Sample CGF and COE Files – Multiple Memory Blocks

The examples below show the CGF file and COE file generated for a memory consisting of multiple blocks.

Sample CGF File Defining 3 Memory Blocks (multiple.cgf)

```
#version2.0
#memory_block_name=block1
#block_depth=32
#data_width=5
#default_word=00
#default_pad_bit_value=0
#pad_direction=right
#data_radix=2
#address_radix=10
#coe_radix=MEMORY_INITIALIZATION_RADIX
#coe_data=MEMORY_INITIALIZATION_VECTOR
#data=
@0
11
01
@8
11
10
01
@14
01
01
```

```
11
@26
01
11
10
01
#end
#memory_block_name=block2
#block_depth=256
#data_width=12
#default_word=1
#default_pad_bit_value=0
#pad_direction=left
#data_radix=8
#address_radix=2
#coe_radix=MEMORY_INITIALIZATION_RADIX
#coe_data=MEMORY_INITIALIZATION_VECTOR
#data=
@100
253
121
@1101
3
771
@10101
676
531
111
2
@100001
666
35
46
171
@10000011
777
123
001
@10000111
12
@11100000
3
2
5
67
76
#end
#memory_block_name=block3
#block_depth=16
#data_width=6
#default_word=0
#default_pad_bit_value=0
#pad_direction=right
#data_radix=10
#address_radix=8
#coe_radix=MEMORY_INITIALIZATION_RADIX
#coe_data=MEMORY_INITIALIZATION_VECTOR
#data=
@0
```

```
6
@4
11
2
3
44
55
@12
10
#end
```

COE file generated for block #1 of the memory specified by the CGF file multiple.cgf (multiple_block1.coe)

```
MEMORY_INITIALIZATION_RADIX=2;
MEMORY_INITIALIZATION_VECTOR=
11000,
01000,
00000,
00000,
00000,
00000,
00000,
00000,
11000,
10000,
01000,
00000,
00000,
00000,
01000,
01000,
11000,
00000,
00000,
00000,
00000,
00000,
00000,
00000,
00000,
00000,
01000,
11000,
10000,
01000,
00000,
00000;
```

COE File Generated for Block #2 of the Memory Specified by the CGF File multiple.cgf (multiple_block2.coe)

```
MEMORY_INITIALIZATION_RADIX=2;
MEMORY_INITIALIZATION_VECTOR=
000000000001,
000000000001,
000000000001,
000000000001,
000010101011,
000001010001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000011,
000111111001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000110111110,
000101011001,
000001001001,
000000000010,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000110110110,
000000011101,
000000100110,
000001111001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
```

```
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
```

```
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000111111111,
000001010011,
000000000001,
000000000001,
000000001010,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
```

```
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000011,
000000000010,
000000000101,
000000110111,
000000111110,
000000000001,
```

```
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001,
000000000001;
```

COE File Generated for Block #3 of the Memory Specified by the CGF File multiple.cgf (multiple_block3.coe)

```
MEMORY_INITIALIZATION_RADIX=2;
MEMORY_INITIALIZATION_VECTOR=
110000,
000000,
000000,
000000,
101100,
100000,
110000,
101100,
110111,
000000,
101000,
000000,
000000,
000000,
000000,
000000;
```

Chapter 7

The Updates Installer

This chapter describes the Updates Installer tool. The chapter contains the following sections.

Overview

The Updates Installer is a CORE Generator™ utility which allows you to install CORE Generator IP and software updates from the Xilinx™ IP Center. The Updates Installer extracts the files from the IP updates (ZIP or JAR) and deposits them into your Xilinx installation directory.

Features

The following is a list of features for the Updates Installer.

- Easy access from the CORE Generator Tools menu.
- Automatically displays the latest CORE Generator IP Updates available on the Xilinx website.
- Automatically loads the URL for each update package selected from the available packages list.
- Performs dependency checks and alerts you to packages that need to be installed first.
- Handles IP update package download (a URL must be specified), archive extraction, and log file creation.
- Automatically installs updates to the proper location.
- Detects IP and CORE Generator software updates which have already been installed.
- Automatically runs the Get Models utility to extract all HDL simulation models bundled with the downloaded IP package.
- Reports installation success or failure.

Install Package Definition

An Install Package is an archive or file in the ZIP or JAR (JAVA Archive) format which contains files needed by the Updates Installer to install an IP, or set of IP (the installer executables). The Install Package also contains files needed by the CORE Generator™ to support the generation of new cores and may also include updates to the CORE Generator software.

Setting Up your Environment

To take advantage of the Updates Installer, you must ensure that your system can connect to the Internet. If you are operating behind a firewall, you need to configure your proxyHost and proxyPort settings.

The Updates Installer attempts to make an Internet connection when the tool starts up from the Tools menu (**Tools → Updates Installer**) and at various other times during the update procedure. If the connection attempt times out, the following dialog box appears, giving you the opportunity to change your proxy settings.

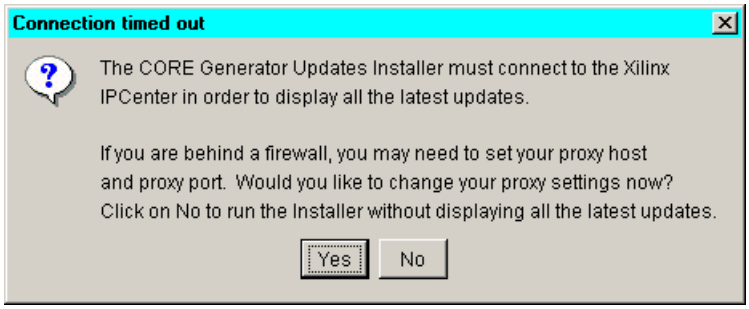


Figure 7-1: **Connection Timed Out Dialog Box**

You can set the proxy settings using the Proxy settings dialog box, as shown in the following figure.

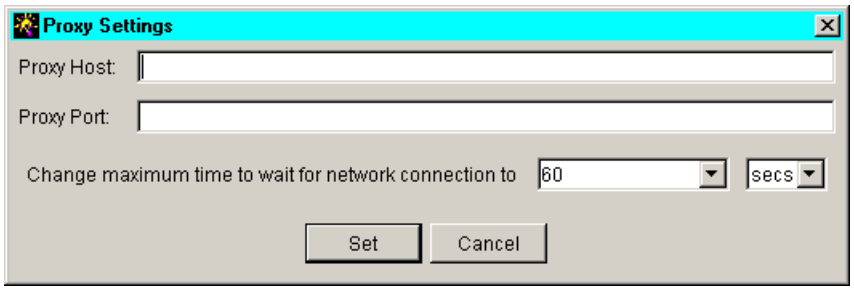


Figure 7-2: **Proxy Settings Dialog Box**

Proxy Settings

The Proxy Settings Dialog box contains the following fields.

- **Proxy Host**

  Contact your system administrator for the name of your Proxy Host.

- **Proxy Port**

  Contact your system administrator for the number of your Proxy Port.

- **Maximum time to wait for network connection**

  This is the amount of time for which the Updates Installer will attempt to make a connection before asking whether you want to change your proxy settings or stop trying.

  The default time your system waits before timing out may be shorter than the value specified here. You cannot use this setting to make your system wait longer for a connection than it is allowed to.

If the Updates Installer cannot make an Internet connection, you can still use the tool to install already downloaded packages. However, you will not be able to access information on what IP Updates packages are currently available or take advantage of the automated Web download process. You will need to download the IP Update archive(s) through your web browser or via FTP, and the IP update must be in the ZIP or JAR file format (TAR files are not supported). Once the archives have been downloaded, you can install them using the Updates Installer by specifying the location of the downloaded archives.

Web Browser Location

The Updates Installer uses your web browser to display any online documentation associated with an update package.

To specify the location of your web browser, follow these steps.

1. In the CORE Generator window, select **File** → **Preferences**.
2. In the **Location of Web Browser** field, enter the appropriate path to your web browser.

User Registration

To download IP Update packages using the Updates Installer, you must be registered with the Xilinx website. To register, follow these steps.

1. Access the Xilinx website, http://www.xilinx.com, through your web browser.
2. Click the **my profile** link at the lower lefthand corner of this web page.
3. Click the **Create an Account** button and complete the registration questions.

Once you are registered, you can access IP updates when you enter the Xilinx website using the Updates Installer.

You may also need to be registered at the IP Update lounge to access a given IP package. For example, to access the standard CORE Generator™ updates, access the registration link at this location:

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

Required Inputs for IP Update Packages

The Updates Installer requires information on the location of each IP update package it must install. You can specify location information in either of these ways:

- A URL pointing to the zip/jar archive corresponding to the update
- A fully qualified or relative path to the zip/jar file on your local hard drive

Note: Use this option if you already downloaded an IP Update to your local hard drive, but would like to have Updates Installer install the downloaded package.

Installing Cores using the Graphical User Interface

The Updates Installer GUI (Graphical User Interface) allows centralized access to packages available for installation and to packages already installed.

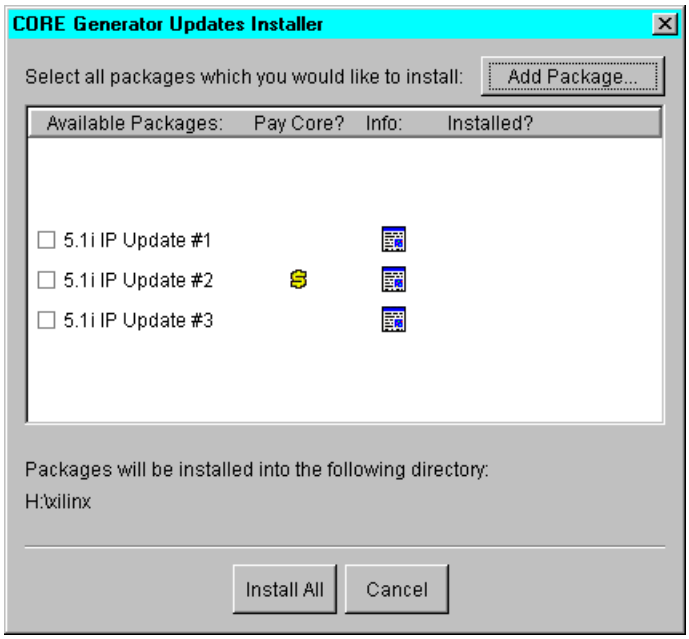


Figure 7-3: **Updates Installer GUI**

The Selection Pane

The center region of this GUI displays a list of packages that are available for installation. Each row corresponds to a single install package and is divided into these fields:

- **Available Packages**

  Allows you to select which package you want to add to the Install Queue.

- **Pay Core?**

  Indicates whether this is a pay core.

- **Info:**

  Provides a link to online documentation for the package.

- **Installed?**

  Indicates whether the corresponding package is already installed.

If your proxy settings are not configured correctly, or if you are unable to connect to the Xilinx™ IP Center, the selection pane will not display in the GUI. If this occurs, you must manually enter the URL or the explicit path to the packages you wish to install.

Selecting Packages to Install

To install packages using the Updates Installer GUI, follow these steps.

1. If you wish to add a JAR or ZIP file to the list displayed in the GUI, click the **Add Package** button, browse to the location of the package you wish to install in the Choose Install Package dialog box, and click **Add to List**.
2. In the Updates Installer GUI, click the check boxes next to the Install Packages you want to install.
3. To install the selected packages, click the **Install All** button.

Note: The CORE Generator™ application will close and a confirmation dialog box will display when the install is complete. The exact amount of time of this process will depend on the size and number of packages you are installing.

Running Get Models

As part of the installation process, the Install Cores Tool automatically runs the Get Models tool after the extraction process is complete for each IP package. (Certain packages contain specific instructions indicating that Get Models should not be run.) The confirmation dialog that appears at the end of the extraction process will indicate when Get Models is invoked. Get Models is run once for each package that is installed.

Get Models extracts the HDL behavioral models of the cores it has installed and copies them to $XILINX/verilog/src/XilinxCoreLib and $XILINX/vhdl/src/XilinxCoreLib, respectively. It also updates analyze order files ($XILINX/verilog/src/XilinxCorelib/verilog_analyze_order and $XILINX/vhdl/src/XilinxCoreLib/vhdl_analyze_order) which specify the order in which the simulation models should be compiled for pre-compiled HDL simulators.

Appendix A

Get Models

This appendix describes the GetModels program. The appendix includes the following sections:

GetModels Overview

The `GetModels` program is a tool that is sometimes needed to extract the Verilog™ or VHDL behavioral models embedded within a user's CORE Generator™ System installation. GetModels extracts these models to a single, central location. It also generates a file (verilog_analyze_order or vhdl_analyze_order) for the CORE Generator Verilog and VHDL libraries. The file specifies the order in which the models must be compiled for compiled HDL simulators.

In the 3.1i and later releases of the Xilinx™ software CDs, the HDL functional simulation libraries for CORE Generator cores are provided pre-extracted in the $XILINX/vhdl/src/XilinxCoreLib and $XILINX/verilog/src/XilinxCoreLib directories. Since new models are also provided pre-extracted with any 6.1i IP updates you may subsequently install, you do not usually need to run GetModels. GetModels needs to be run only if the install extraction fails, or if you install third party IP modules which have been configured so that they are catalogued, elaborated and delivered by the CORE Generator.

In a Xilinx software installation, CORE Generator models may exist in the CORE Generator tree ($XILINX/coregen) in either JAR archive or ASCII 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 MTI ModelSIM, Synopsys VCS and Cadence NC-Verilog), extracting the behavioral models to a single library directory allows them to be conveniently analyzed by the Verilog or VHDL simulator.

You can also use GetModels to extract individual behavioral models for specific CORE Generator modules to your project directory if necessary.

GetModels can either be invoked from the CORE Generator™ **Tools** menu, or from the command line by typing the **`get_models`** command line (see "Command line Syntax"). The **Tools** menu dialog box for GetModels is shown following.

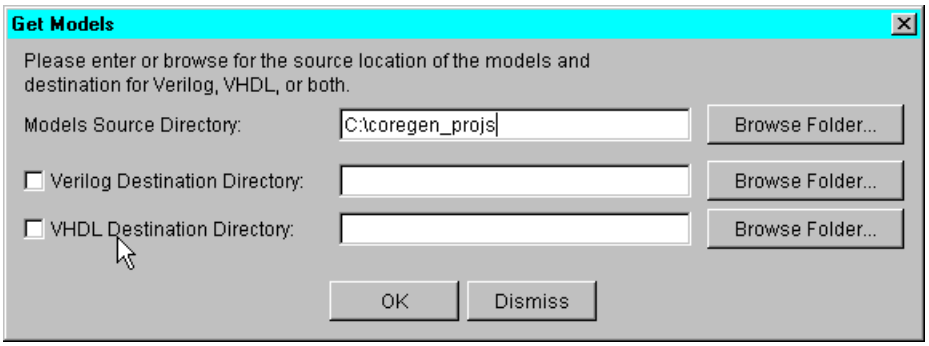


Figure A-1: **GetModels Dialog Box**

The syntax for running GetModels in command line mode is as follows:

Command line Syntax

```
get_models [-h | -help] -vhdl | -verilog
[-dest destination_directory]
```

Required Parameters

`-vhdl|-verilog`

Using these parameters extracts Verilog™ or VHDL behavioral models only and creates the appropriate verilog_analyze_order or vhdl_analyze_order file. You must specify either the **`-verilog`** or **`-vhdl`** option when running the get_models command.

Optional Parameters

`-dest` *destination_directory*

Use this parameter as the target location for the XilinxCoreLib library and any VendorCoreLib libraries, if applicable. The metacharacters "."and ".." are supported.

For most standard installations, only a XilinxCoreLib subdirectory is created by get_models in the specified directory. The default location of *destination_directory* when the **`-verilog`** option is specified is $XILINX/verilog/src. The default location of *destination_directory* when **`-vhdl`** is specified 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 GetModels are the CORE Generator™ behavioral models installed in your Xilinx CORE Generator System repository. The models exist in either of the following two formats:

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

Note: It is usually not necessary to specify the path to these inputs when running GetModels. 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 to a library named XilinxCoreLib or *Vendor*CoreLib. Information on the GetModels extraction run is logged to getmodels.log.

For Verilog compiled simulators, the required compilation order is indicated in a file named verilog_analyze_order. The order specified in this file must be followed if you wish to reference this library using the **-y** Verilog simulator option.

VHDL simulators require models to be compiled from the bottom-up using lower level blocks before higher level blocks. There can be more than one compile order that meets this bottom-up order requirement. The order specified in the vhdl_analyze_order file is just one such combination.

Appendix B

Configuration Files and Global Preferences

This appendix describes the following features:

- "CORE Generator Configuration Files"
- "Global Preferences"

CORE Generator Configuration Files

The following sections describe some CORE Generator™ configuration files.

coregen.prj File

The coregen.prj file is the CORE Generator project file. The coregen.prj file is automatically created in the project directory 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 file formats. To be a valid CORE Generator project, the project directory must contain a valid 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.

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.

.coregen.prf File

The .coregen.prf file is the Xilinx™ CORE Generator preferences file for UNIX™ workstations. This is an ASCII option settings file that records various user specific preference 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 a .coregen.prf in your home directory on UNIX platforms. During start-up, and after any optional coregen.ini file is read (Workstations

only), CORE Generator™ System searches your home 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, as in the case of a first-time user, the various preference values take on their hard coded default values.

The first time you start up the CORE Generator System, no .coregen.prf preference file exists. It is created the first time you exit out of the CORE Generator System based on settings you have specified during a project session. The file is automatically written to your home directory.

On Windows platforms, user preferences are stored in the Windows registry.

Supported Preference File Properties

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

Table B-1: **Preference File Table**

Field	Value	Description
AlwaysOpenLastProject	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=8).
LastProject	*<path_to_last_project>*	Path to last project you opened.
LastProject*<n>*	*<path_to_project_n>*	Path to project n in list of previously open projects.
OverwriteFilesDefault	False	Prompt user before overwriting files during elaboration.
	True	Automatically overwrite design files during elaboration.
WebBrowser	*<path_to_web_browser>*	Set fully qualified path to your web browser.
PDFViewer	*<path_to_pdf_viewer>*	Set fully qualified path to your Acrobat™, Netscape™, or other PDF viewer.

Preference File Example

The following is an example of a workstation preference file:

```
#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
```

Global Preferences

A global preference is a persistent property that is user specific and applies to all projects. These are set either through the Preference Options dialog box available from the main CORE Generator application GUI, or by using SET commands in a command file. The persistent values are stored in a file named .coregen.prf in $HOME on UNIX platforms. On WIN / NT platforms the persistent values are stored in the Windows Registry in this location:

HKEY_CURRENT_USER\Software\Xilinx\Coregen\Preferences

The following table describes the Global Preferences commands, values, and their functions:

Table B-2: **Global Preferences**

Property	Values	Description
AlwaysOpenLastProject	True | False	If True, upon invoking CORE Generator™, the last project used will be opened automatically. Default is False.
CloseIPGUIAfterGeneration	True | False	If True, the IP customization GUI for a core will close automatically after the core is generated. If False, the GUI will remain open after the core is generated. The default is False.
DefaultRepositories	<*repository_list*>	When creating a new project or converting a deprecated project use these IP repositories for configuring the project initially. The default is the directory $XILINX/coregen/ip/xilinx.
DisplaySupported	True | False	Cores that do not support the family selected for the current project are normally displayed as grayed out in the main CORE Generator GUI. If this is set to True, unsupported cores are not displayed at all.

Table B-2: **Global Preferences**

Property	Values	Description
DumpCoreStatistics	True | False	The Core Viewer, when invoked, displays statistics on resource usage as well as the core footprint. If set to True, these resource numbers will also be written out to the XCO file of the generated core. Can be set by batch command only.
LastNProjects	<*Integer*>	Number of last accessed projects to be stored in the Preference Manager. Default is 8.
OverwriteFilesDefault	True | False	If a new project is created and the project property OverwriteFiles is not set, then the overwrite behavior will be determined by this preference. Default is False.
PDFViewer	<*PDF_viewer_path*>	Set the path to the PDF / Acrobat file reader of choice. Default is the path to *acroread*.
ProxyHost	<*host_name*>	Specify the proxy host to be used with the selected web browser.
ProxyPort	<*port_id*>	Set the proxy port of the proxy host to be used with the selected web browser.
WebBrowser	<*browser_path*>	Set the path to the web browser of choice. Default is the path to *netscape*.

Appendix C

Troubleshooting the CORE Generator System

This appendix contains solutions and resources for using the CORE Generator™ System. The appendix contains the following sections:

Finding Solutions

Following are some methods of finding solutions to CORE Generator problems:

- 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 (**Project** → **New**) 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 to add a **-d** (debug mode) option to the java.exe command line in coregen.bat. The **-d** option causes the CORE Generator System to display a detailed report of all data files being loaded, miscellaneous operations, and debug-related information.

Additional Resources

Following are some additional CORE Generator™ 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. 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, see the Xcell Journal Online at the following website:

  http://www.xilinx.com/publications/xcellonline

- For Solution Records related to a specific core, click the **Web Links** tab in the core's customization GUI. Then click **Solution Records for this Core**.

AllianceCORE Modules

Contact the appropriate third party AllianceCORE™ provider as indicated on the CORE Generator data sheet 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/support/techsup/tappinfo.htm.

Index

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X