Alliance Series 2.1i Quick Start Guide

About This Manual

This manual provides an overview of the Alliance Series 2.1i Software, including a basic tutorial. There are also instructions for how to configure your third-party interface tools to work with the Alliance software flow. This manual is targeted for the user who has already installed their software and online documentation, and set up their user environment variables.

Other publications you can consult for related information include the *Design Manager/Flow Engine Guide* and the *Alliance Release Notes and Installation Guide*.

Additional Resources

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

Resource	Description/URL
Tutorial	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://support.xilinx.com/support/techsup/tutorials/index.htm
Answers Database	Current listing of solution records for the Xilinx software tools Search this database using the search function at http://support.xilinx.com/support/searchtd.htm
Application Notes	Descriptions of device-specific design techniques and approaches http://support.xilinx.com/apps/appsweb.htm

Resource	Description/URL
Data Book	Pages from *The Programmable Logic Data Book*, which describe device-specific information on Xilinx device characteristics, including read-back, boundary scan, configuration, length count, and debugging http://support.xilinx.com/partinfo/databook.htm
Xcell Journals	Quarterly journals for Xilinx programmable logic users http://support.xilinx.com/xcell/xcell.htm
Tech Tips	Latest news, design tips, and patch information on the Xilinx design environment http://support.xilinx.com/support/techsup/journals/index.htm

Manual Contents

This manual covers the following topics.

- Chapter 1, “Introduction” introduces the various features of the Xilinx software.
- Chapter 2, “Implementation Tools Tutorial” provides a tutorial on the Xilinx design flow.
- Chapter 3, “Using the Software” looks in-depth at the capability and flexibility of the Alliance software tools.
- Appendix A, “Alliance FPGA Express Interface Notes,” covers how to install and start using FPGA Express for the Alliance 2.1i Software.
- Appendix B, “Mentor Graphics Interface Notes,” covers how to set up the Mentor Graphics interface and associated libraries.
- Appendix C, “Xilinx Synopsys Interface Notes,” covers how to set up the Xilinx Synopsys Interface (XSI) and associated libraries.
- Appendix D, “Viewlogic Interface Notes,” covers how to set up the Viewlogic interface and project libraries.
- Appendix E, “Using LogiBLOX with CAE Interfaces,” covers how to set up the LogiBLOX interface and associated libraries
- Appendix F, “Instantiated Components,” includes a listing of the components most frequently instantiated in synthesis designs.

- Appendix G, “Alliance Constraints,” describes the most common constraints you can apply to your design to control the timing and layout of a Xilinx FPGA or CPLD.
- Appendix H “Configuring Xprinter,”provides configuration details for Workstation users so they can print from Xilinx GUI applications.
- Appendix I, “Glossary,” contains definitions and explanations for terms used in the Quick Start Guide.

Conventions

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

Typographical

The following conventions are used for all documents.

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

  ```
  speed grade: -100
  ```

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

  `rpt_del_net=`

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

  `File` → **`Open`**

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

    `edif2ngd` *design_name*

  - References to other manuals

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

  - Emphasis in text

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

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

  `edif2ngd` [*option_name*] *design_name*

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

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

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

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

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

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

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

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

Online Document

The following conventions are used for online documents.

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

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

Contents

Chapter 1

Introduction

This chapter contains basic information about the Alliance 2.1i Software and its components, along with listing the Xilinx Devices that are supported by the software.

For complete information about the new features of this software release, refer to the “What’s New” file included on your Alliance Implementation Tools CD-ROM.

This chapter contains the following sections.

- “Supported Devices”
- “Supported Netlists”
- “Xilinx Development System Tools and Features”
- “Software Manuals and Online Help”
- “EDA and Third Party Interface Support”
- “Software Installation and Licensing”
- “Support and Services”

Note: Complete software installation information is located in the *Alliance 2.1i Release Notes and Installation Guide.*

Supported Devices

The Alliance 2.1i Software supports the following device families.

- XC3000A / L
- XC3100A / L
- XC4000E / L / EX / XL / XV / XLA
- XC5200

- XC9500/X
- Spartan/XL/II
- Virtex/E

Refer to *The Programmable Logic Data Book* for more information on these devices. The online version of the *Data Book* is at http://www.xilinx.com/partinfo/databook.htm.

For updated information regarding speed grades and package support, search the Xilinx Answers Database and the latest Application Notes. You can search Xilinx Technical Documentation at http://www.xilinx.com/support/searchtd.htm.

Supported Netlists

You must use the Xilinx Unified Libraries to create your designs. Refer to the Xilinx *Libraries Guide* for a list of components. The following table lists the netlist formats supported by the Xilinx software.

Netlist Format	Variations
EDIF	SEDIF, EDN, EDF, EDIF
XNF	SXNF, XFF, XTF, XNF

Xilinx Development System Tools and Features

This section lists the tools and the main features of the Xilinx software. The tutorial in this manual provides a brief overview of how to use these software tools.

For detailed information on using the following Xilinx GUI tools, refer to the appropriate online software manual.

Table 1-1 Xilinx Software Tools

Feature	Description
Design Manager	Top level software module in the Xilinx Development System. The Design Manager provides access to all the tools you need to read a file from a design entry tool and implement it in a Xilinx device.
Flow Engine	Displays and executes all the steps needed to implement a Xilinx design, including translating design netlists; mapping logic to CLBs; placing and routing designs; creating a configuration file for downloading to a device; creating static timing reports; and creating timing simulation netlists in VHDL (Vital), Verilog, EDIF, or XNF.
LogiBLOX	Graphical tool used to create high-level modules, such as counters, shift registers, and multiplexers.
CORE Generator	The CORE Generator system has been integrated into the Alliance 2.1i Software interface. The CORE Generator GUI tool generates and delivers parameterizable cores optimized for Xilinx devices.
Floorplanner	Graphical tool used to control the placement of your design into a target FPGA using a “drag and drop” paradigm with the mouse pointer.
Constraints Editor	Graphical tool used after running NGDBuild to add timing constraints and I/O pin locations.
FPGA Editor	Graphical tool used to display and configure your designs before or after placing and routing.
Hardware Debugger	Used to download your design to a device, verify the downloaded configuration, and display the internal states of the programmed device.
PROM File Formatter	Creates files for serial or byte-wide configuration PROMs. Three formats are available: MCS, EXO, and TEK. The HEX format is also supported for microprocessor-based configuration.

Table 1-2 Xilinx Software Features

Feature	Description
Timing Specification Performance	Xilinx software supports timing-driven placement and routing.
Multi-Pass PAR	The place and route (PAR) software allows multiple place and route iterations on a single machine, a UNIX™ network, or on multiple machines running in parallel. This feature provides optimum performance and efficiency, utilizing CPU time to achieve faster design results.
Re-Entrant Routing	Re-entrant routing skips placement and routes your design. Routing begins with the existing placement and routing left in place.
Guide for Incremental Design Changes	You can select a previously mapped, routed, or fitted implementation revision to use as a guide for implementation.

Software Manuals and Online Help

Xilinx provides software user manuals and online help for its GUI tools and associated EDA software interfaces. You can access the online help from the **`Help`** → **`Help Topics`** pull-down menu option of each software tool. The following sections provide more information about accessing and using the Software Manuals Online.

New Features for Software Manuals

The Xilinx Software Manuals are now provided on the Web and on your Documentation CD-ROM. You can install the software manuals locally, read them from the CD-ROM, or read them on the Xilinx Web site. Easy print options are also available. The web-compatible documentation includes powerful search functions is viewed using your own Java compatible internet browser.

Note: For best performance, Xilinx recommends you use version 4.0 or higher of either Netscape Navigator tm or Microsoft Internet Explorer tm browsers.

Software Manuals On the Web

The 2.1i Series Software Manuals are accessible from the Xilinx Support Web site at the following location. Bookmark this link for easy future use, or add the URL to your Favorites list.

http://support.xilinx.com/support/sw_manuals/2_1i/index.htm

Online Help for Software Manuals

Online help instructions for reading, browsing, and searching the online manuals are available through the web browser interface. Click the **`Help`** button in the upper left-hand corner of the documentation viewer browser window to access the help topics.

Note: The new web-compatible documentation will appear the same whether you access the manuals locally (from the CD-ROM) or through the Web site.

Installing Software Manuals

Complete directions for installing the online Software Manuals and the Alliance Implementation tools software are located in the *Alliance 2.1i Release Notes and Installation Guide.*

In order to use the new documentation viewer, you must have a Java-enabled web browser installed on your system or network. During Alliance Implementation Tools installation, you can specify the path to your current web browser or install Netscape Navigator 4.0.5.

You have two options for installing the Software Manuals.

- Install the Manuals locally or to a network location
- Access the Manuals through the web

Xilinx recommends that you install all of the online manuals locally for fastest access. If you install the manuals, you will be able to access them using the **`Help`** → **`Online Documentation`** pull-down menu command in the Design Manager and other Xilinx GUI tools.

If you wish to save space on your local drive, you may choose to access the manuals through the web instead of installing them. For information about printing the software manuals, see the “Printing Software Manuals” section.

When you read the Software Manuals for the first time, a “Java Security” window may appear. This window is requesting additional permissions for Docsan, the Java application used to view the software manuals. Select the **`Grant`** or **`Yes`** button to allow Docsan to access your hard disk.

Warning: You must grant this permission to view the software manuals. For your security, the Docsan Java applet is digitially signed by Sidana Systems, Inc. using a certificate from Verisign.

Printing Software Manuals

You can now print the entire Xilinx software manuals with the graphics and text inline. PDF format files of each of the software manuals are provided on the 2.1i Software Documentation CD-ROM.

Printing PDF Files

To access and print using the PDF files, use the following steps.

1. Verify that you have Adobe Acrobat Reader (version 3.0 or above) installed on your network or local area.

   You can install Acrobat from the Alliance 2.1i Implementation tools software CD-ROM by selecting the Core Generator option.

2. Start Acrobat Reader

   - Unix users

     Run the following command to start this tool.

     /**`usr`**/**`bin`**/*path_to_directory*/**`acrobat`**

     Path_to_directory is the directory where your Acrobat program files are located.

   - PC users

     Select **`Start`** → **`Programs`** → **`Acrobat Reader`**

3. Insert the Xilinx 2.1i Software Documentation CD-ROM into your drive.

4. Access your CD-ROM directory.

   - Unix users

Enter /**usr**/**bin**/*path_to_CD-ROM_directory*, where *path_to_CD-ROM_directory* is your mounted CD-ROM drive.

Enter the **`ls`** command to view the contents of the CD-ROM directory.

- PC users

  Select **Start** → **Programs** → **Windows Explorer** and select your CD-ROM drive to display the contents of the CD-ROM.

5. Open the **Print** directory or folder from the CD-ROM contents. This directory contains the PDF files.
6. Select a book and open it in Acrobat.

   The following table lists the book titles and their corresponding PDF file names.

Table 1-3 List of Alliance 2.1i Software Manuals

Manual Title	PDF File Name
Xilinx/Concept-HDL Interface Guide	docchdl.pdf
Constraints Editor Guide	cst_edit.pdf
CPLD Schematic Design Guide	sdg_alli
CPLD Synthesis Design Guide	syn_cpld
Design Manager/Flow Engine Guide	dmfe.pdf
Development System Reference Guide	dev_ref.pdf
FPGA Editor Guide	fpedit.pdf
Floorplanner Guide	fplan.pdf
Hardware Debugger Guide	hdebug.pdf
Hardware User Guide	huguide.pdf
JTAG Programmer Guide	jtag.pdf
Libraries Guide	libguide.pdf
LogiBLOX Guide	lblox.pdf
Mentor Graphics Interface Guide	mentor.pdf
PROM File Formatter Guide	prom_fmt.pdf
Alliance Series 2.1i Quick Start Guide	docaqsg.pdf
Synthesis and Simulation Design Guide	gensim.pdf

Table 1-3 List of Alliance 2.1i Software Manuals

Manual Title	PDF File Name
Synopsys Synthesis and Simulation Design Guide	xsisyn.pdf
Xilinx/Synopsys Interface Guide	xsi_int.pdf
Timing Analyzer Guide	timing.pdf
Viewlogic Interface Guide	vlifg.pdf

7. Print the book using the **`File`** → **`Print`** command from the Adobe Acrobat reader window.

Printing from the Online Document Viewer

You can print individual pages of the Software Manuals directly from your internet browser window. For example, Netscape users would use the **`File`** → **`Print Frame`** menu option from their browser window. (Make sure that you have clicked on the right-hand book view frame in order to select it for printing.)

Xilinx recommends that you use the online books for quick information access and searching, and the PDF files for best print quality. Graphics in the web-based manuals are not inline and will not print automatically. They are also sized for optimal online viewing and may not fit on a printed page.

If you do not have access to your Documentation CD-ROM, you can also access the Software Manuals in PDF format on the Web. You can FTP each manual to your local area.

Note: This process is slower than accessing the PDF files directly from your CD-ROM.

EDA and Third Party Interface Support

The Alliance 2.1i software supports various third party interfaces. For the most current information on the latest vendor version support in the Alliance EDA partner program, refer to http://www.xilinx.com/programs/alliance/alligen.htm.

Software manuals for EDA interface users are provided on your software documentation CD-ROM. You can also access the manuals on the web, from support.xilinx.com.

Several Appendixes in this manual provide information on how to set up your EDA vendor tools to interface with Alliance 2.1i software.

Software Installation and Licensing

Complete software installation instructions are located in the *Alliance 2.1i Release Notes and Installation Guide.*

When you install the software, you will be asked to provide your CD-KEY. This key instructs the installation program to load the software package that you purchased. Your CD-KEY, which typically starts with the letters “AB” or “AS,” is located on a sticker on the back of the CD-ROM holder.

You will also be asked to provide your software serial number during software installation. This number helps the technical and customer support team assist you more efficiently. This also ensures that you will be eligible to receive free software updates as they become available.

For new Xilinx users, this “SN” number is located on a sticker on the back of your CD-ROM holder.. If you are a current Xilinx customer, your serial number will appear on the mailing label of your shipping package.

You do not need a license to run the Alliance 2.1i software. However, you must be a registered user in the Xilinx Customer Service database in order to receive the full benefits of your customer and technical support. If you have not registered your sofware, you can do this online at http://www.support.xilinx.com.

At the end of installation, new Xilinx software users should select the “Online Registration” option to Register on the Web. Alternatively, you could fill out the Xilinx registration card and fax or mail it to your Customer Service location. You only need to register your software once. This will ensure that you receive future updates (during your warranty period) and future product information.

Support and Services

This section provides information for contacting your technical support and customer service representatives.

Technical Support

If you experience problems with your software installation or operation you can look for solutions and answers at http://support.xilinx.com. The Xilinx technical support web site also provides forms for easily submitting your technical questions by e-mail. To access these forms, go to the "Services" area of support.xilinx.com and click the "Open New Case" link.

If you need additional support, contact the Xilinx Technical Support hotline by phone or fax. When faxing inquiries, provide your complete name, company name, and phone number, along with the software version you are using.

Location	Telephone	Facsimile (Fax)
North America	1-408-879-5199 1-800-255-7778	1-408-879-4442
United Kingdom	44-1932-820821	44-1932-828522
France	33-1-3463-0100	33-1-3463-0959
Germany	49-89-93088-130	49-89-93088-188
Japan	local distributor	local distributor
Korea	local distributor	local distributor
Hong Kong	local distributor	local distributor
Taiwan	local distributor	local distributor
Corporate Switchboard	1-408-559-7778	

Customer Service

This section provides information for contacting your local Xilinx Customer Service representative. Contact your local distributor for international countries not listed.

The offices for the US and Canada are open Monday through Friday from 8:00 am to 5:00 pm Pacific time.

The European offices are open Monday through Friday from 9:00 am through 5:30 pm, United Kingdom time. These offices are English-speaking only.

Country	Telephone	Facsimile
United States and Canada	1-800-624-4782	408-559-0115
United Kingdom	01932-333550	01932-828521
Belgium	0800 73738	
France	0800 918333	
Germany	0130 816027	
Italy	1677 90403	
Netherlands	0800 0221079	
Other European Locations	(44) 1932-333550	(44) 1932-828521
Japan	81 3 3297 9153	81 3 3297 9189

Chapter 2

Implementation Tools Tutorial

This chapter contains the user instructions for a tutorial that covers many functions of the Alliance 2.1i Implementation Tools. Using this tutorial is a good way for a new user to learn how the Alliance design flow works with basic designs.

Note: An updated version of this tutorial will be available after July 7th, 1999 from the Xilinx Support web site as well as on the AppLINX CD. The web site location is http://support.xilinx.com/support/techsup/tutorials/index.htm. Contact your local sales representative to obtain a copy of the AppLINX CD.

This chapter contains the following sections.

- "Installing the Tutorial Files"
- "Step 1: Creating an Implementation Project"
- "Step 2: Specifying Options"
- "Step 3: Translating the Design"
- "Step 4: Using the Constraints Editor"
- "Step 5: Mapping the Design"
- "Step 6: Using Timing Analysis to Evaluate Block Delays After Mapping"
- "Step 7: Placing and Routing the Design"
- "Step 8: Evaluating Post-Layout Timing"
- "Step 9: Creating Timing Simulation Data"
- "Step 10: Creating Configuration Data"
- "Step 11: Using the PROM File Formatter"

Installing the Tutorial Files

This tutorial demonstrates the Alliance Series Design Implementation Tools flow. The front end design has already been compiled for you in an EDA Interface tool and is described by an EDIF Netlist File (EDF). For a listing of EDA Interface tutorials, please reference the Xilinx Support area referenced at the beginning of this chapter.

This tutorial passes an input netlist from the front end tool to the back-end Alliance Series 2.1i Design Implementation Tools, and then incorporates placement constraints through a User Constraints File (UCF). Timing constraints will be added on later through the Constraints Editor.

The tutorial design, titled “Watch,” is designed to perform like a track coach’s stopwatch. There are two inputs to the system (RESET and SRTSTP). The configuration clock on the device is used as a ten hertz (HZ) clock signal. Three seven-bit outputs are generated by this system for output to three seven-segment LED displays.

Before proceeding to Step 1 in the tutorial, create a working directory with the tutorial files as follows.

1. Create an empty working directory named Watch.
2. Copy the following files from the $XILINX/userware/tutorial/qstart/ directory into to your newly created working directory.

The following table lists the relevant file names and a description.

File Name	Description
watch.edn	Input netlist file (EDIF)
tenths.ngc	LogiBLOX implementation file
watch.ucf	User constraints file

Note: In order for the /userware/tutorial/qstart directory to be present in your root Xilinx directory, you must first install the Userware Tutorial files from the Alliance Series Design Implementation Tools CD-ROM.

Step 1: Creating an Implementation Project

The Design Implementation Tools are organized under a single program called the Design Manager. The Design Manager helps you manage the design flow process by keeping track of design versions and the implementation revisions within each version. The Design Manager also provides access to the entire suite of Xilinx implementation tools needed to complete a design.

While the Design Manager manages your Xilinx design, the Flow Engine actually implements it. The Flow Engine is closely integrated with the Design Manager and shares many of the same menus and dialog boxes.

To begin, use the following steps to create an implementation project.

1. On a workstation, enter the following to start the Design Manager.

   `xilinx &`

   On a PC, select the following to start the Design Manager.

   `Start` → **`Programs`** → **`Xilinx`** → **`Design Manager`**

   When you open the Design Manager for the first time, you must create a new project for your design. A project includes all design versions, implementation revisions, reports, and any other Xilinx data created while you work with a design.

   The Design Manager graphically displays information about these items in the project view. When you create a new project, you specify a design to open and a directory for the project. You can create as many projects as you want, but you can only work with one at a time.

2. Select **`File`** → **`New Project`** from the Design Manager menu to create a new implementation project for the tutorial design. The New Project dialog box appears.

   The fields of this dialog box are described in the following table.

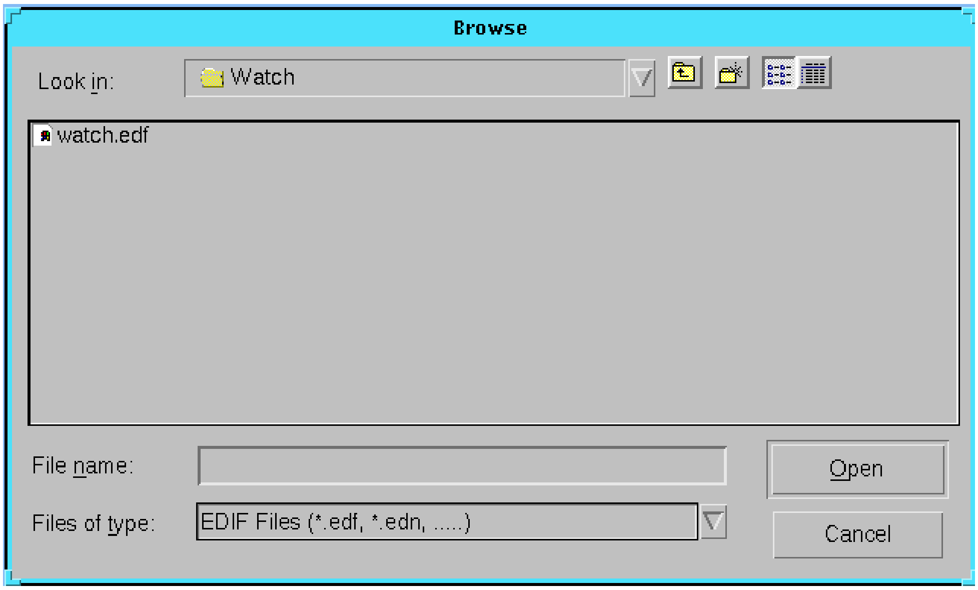
Table 2-1 New Project Dialog Box Fields

Field	Description
Input Design	Top level netlist file containing the design definition

Table 2-1 New Project Dialog Box Fields

Field	Description
Work Directory	Directory used to store the implementation data created as the design is compiled
Comment	Enter any optional notation for the design in this field

3. To specify your input design, click the Browse button to the right of the Input Design field. The Browse dialog box appears as shown in the following figure.



File name:

Files of type: EDIF Files (*.edf, *.edn, ...)

Figure 2-1 Browse Dialog Box

4. Select the appropriate file type from the drop-down list in the Files of Type field. For this tutorial design, EDIF is selected.

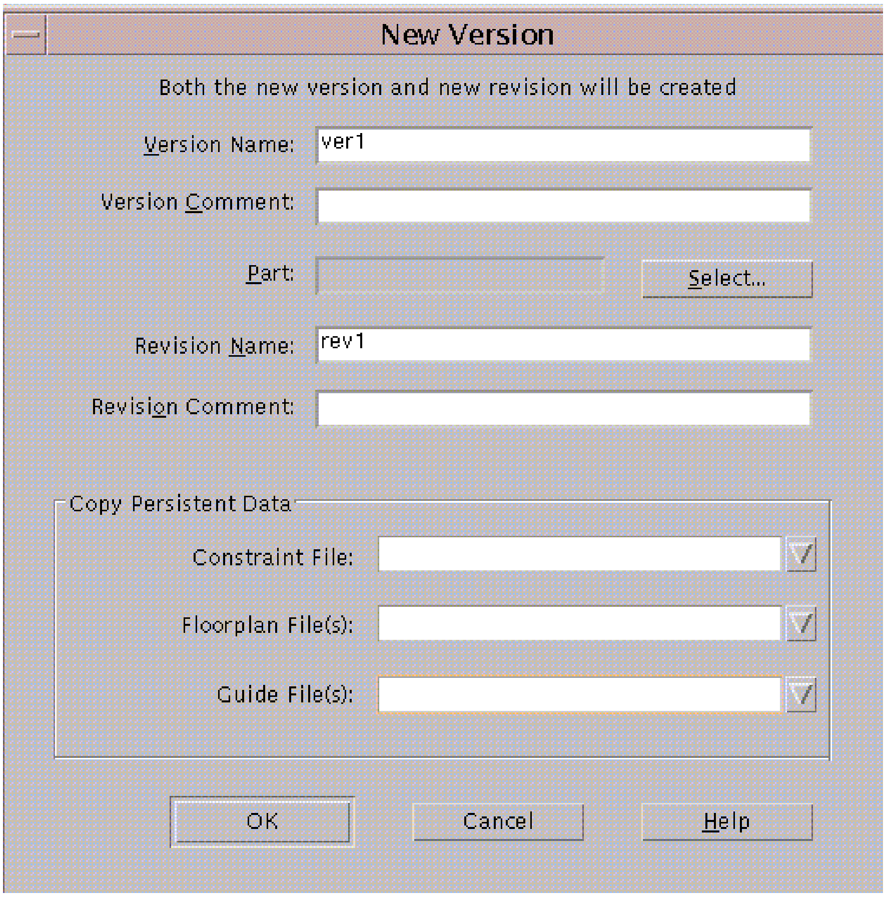
5. Select the Watch design file. The file name appears in the File Name field. Click **Open**.

   The Browse dialog box closes and the New Project dialog box is updated to include the specified input netlist. By default, the Work Directory field is set to the directory containing the input design. If preferred, you can set this to another directory. Because the files were previously copied to the Watch directory, this directory is used for the implementation project and resulting output files.

6. In the Comment field, enter the following.

   `-tutorial`

7. Click **OK** to close the New Project dialog box. The New Version dialog box appears, as shown in the following figure.



New Version

Both the new version and new revision will be created

Version Name: ver1

Version Comment:

Part:

Revision Name: rev1

Revision Comment:

Copy Persistent Data

Constraint File:

Floorplan File(s):

Guide File(s):

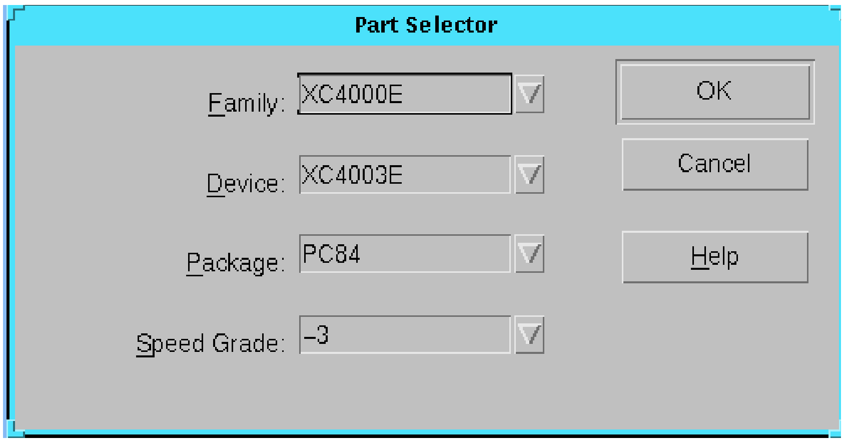
Figure 2-2 New Version Dialog Box

When you initially creating your project, the New Version dialog automatically appears to allow you to enter the information necessary to define the new design version. Furthermore, any time that your input netlist changes due to a change made within your front-end tool, you are prompted by the Design Manager to generate a new design version.

Once a design version is created, you can try different implementation strategies on your design. The data associated with each of these implementation strategies is called an implementation revision. Because a new implementation revision is automatically created when you create a new version, you will see both of these fields already defined in the New Version dialog.

8. By default, the Version Name field shows ver1 as the default version, and the Revision Name field shows rev1 as the default revision. Comments to note options and strategies can be entered in the Version and Revision Comment fields.

9. Click **`Select`** to display the Part Selector dialog box. The Part field will automatically contain a part number if you specified the target device in your design entry tool. Since this field is empty in our example, we must define it.



Part Selector

Family: XC4000E

Device: XC4003E

Package: PC84

Speed Grade: -3

Figure 2-3 Part Selector Dialog Box

10. Use the drop-down lists for the fields in the Part Selector dialog box to enter the Family, Device, Package, and Speed Grade for the design. This design targets an XC4003E-3-PC84. Click OK. The part number appears in the Part field in the New Version dialog box.

11. The Copy Persistent Data heading allows you to specify the copying of constraint, guide, or floorplan data to the new revision that is about to be created. You can choose to copy data from a previous revision or a custom file or choose None if you do not want to copy data. For this tutorial, we will keep all 3 drop-down boxes defined as None.

Note: By default, the Design Manager copies floorplan and constraints file data from the “last” revision. The “last” revision is the bottommost revision in the Design Manager project view. When initially creating a project, the Design Manager copies constraints and floorplan file data (if it happens to exist), from the project directory to the revision directory.

12. Click **OK** in the New Version dialog box.

    The Design Manager loads your design and displays a new design version and implementation revision icon in the project view, as shown in the following figure.

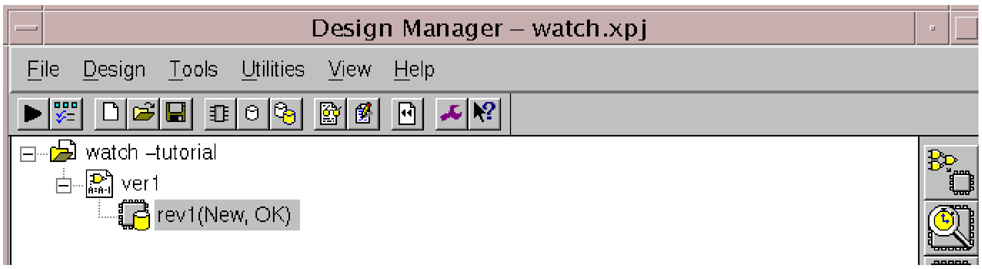


Figure 2-4 Watch Project in Design Manager

Design Manager Status Bar

At the bottom of the Design Manager window is the status bar. The status bar lists the current project, target device, and currently selected version / revision pair. The left-hand portion of the status bar provides help on what is currently selected by your cursor, as shown in the following figure.

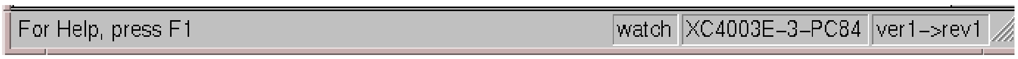

For Help, press F1 | watch | XC4003E–3–PC84 | ver1–>rev1

Figure 2-5 Design Manager Status Bar

Design Manager Toolbox

The toolbox, located on the right side of the Design Manager window, becomes active when a revision is selected. Icons in the toolbox (shown in the following figure) represent the Flow Engine, Timing Analyzer, Floorplanner, PROM File Formatter, Hardware Debugger, FPGA Editor, and JTAG Programmer tools.

Note: The toolbar has drag and drop capability.

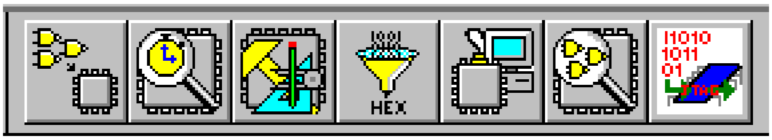


Figure 2-6 Design Manager Toolbox

Step 2: Specifying Options

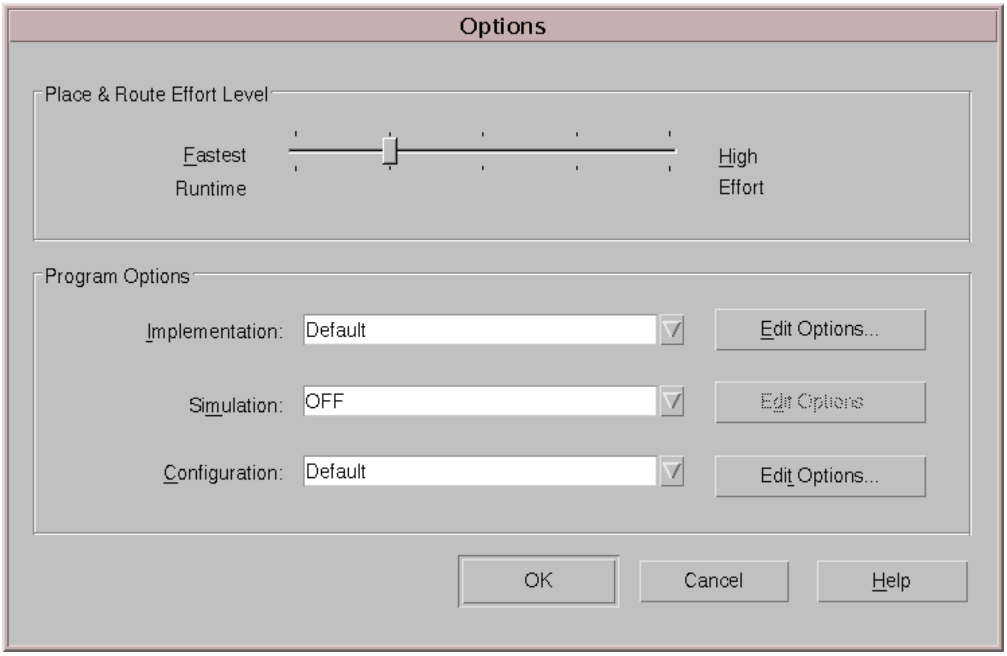
An implementation revision contains data files and reports that are created based on a specific set of implementation strategies. Implementation strategies are defined by specifying a set of options. You can specify options that control how the Flow Engine implements a design, creates timing simulation data, creates netlist files, generates reports, and creates configuration data. The options available depend on the target device family.

You can use the tools to create as many implementation revisions as you want for a design version. For example, if you want to try various implementation strategies on a netlist, several revisions can be created for a single design version. By default, however, the Design Manager recompiles within the current revision.

Within the Design Manager, notice how the project view displays rev1 under the initial version of the watch project. The status of the revision is noted as (New, OK). *New* refers to the state of the design and is updated throughout the tutorial as the different compilation stages are completed. *OK* is the status of the current state and indicates no errors in the design processing.

Use the following steps to specify options for this design.

1. Select **`Design`** → **`Options`** to open the Options dialog box as shown in the following figure.



Implementation: Default

Simulation: OFF

Configuration: Default

Figure 2-7 Options Dialog Box

This dialog allows you to set options used in the implementation, simulation, and configuration flow. Changes made in this dialog box apply to the selected implementation revision. The dialog box above appears if you are targeting an FPGA. A slightly different Options dialog would appear if you were targeting a CPLD.

Select the Help button to read through the information regarding this entire dialog box.

2. Select Edit Options next to the Implementation Program Option.

   The XC4000 Implementation Options dialog box is displayed as shown in the following figure. The implementation options control how the software maps, places, routes, and optimizes a design.

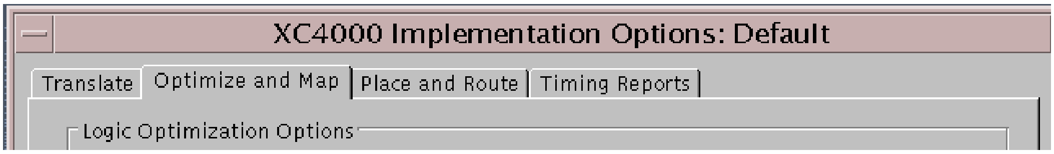

XC4000 Implementation Options: Default
Translate
Optimize and Map
Place and Route
Timing Reports
Logic Optimization Options

Figure 2-8 XC4000 Implementation Options Dialog Box

3. Select the **`Timing Reports`** tab.
4. Select **`Produce Logic Level Timing Report`**. The option to produce a Post Layout Timing Report should already be selected by default. For both reports, select Report Paths in Timing Constraints.

   The timing reports are useful for evaluating design performance. They will be analyzed in detail later in this tutorial.
5. Click **`OK`** to save the Implementation options and return to the Options dialog.
6. Select **`OFF`** from the drop down list next to the Configuration Program Options. This disables the generation of a bitstream for our design. We will revisit this option later once we have completed evaluating the performance of our design.
7. Click **`OK`** to exit the Options dialog box.

As previously mentioned, an implementation revision is created based on a specific set of implementation strategies. In addition to the program options we just set, an implementation strategy is defined by the constraints applied onto the design.

For this design, you were initially asked to copy over a User Constraints File (UCF) into your design directory. Since the Design Manager by default copies over your constraints information into the new revision created, you should be able to open the watch.ucf file found under your newly created rev1 directory. With a text editor, view the location constraints that are specified for this design.

The User Constraints File (UCF) provides a mechanism for constraining a logical design without returning to the design entry tools. However, it requires the user to understand the exact syntax needed to define constraints. On the other hand, the Constraints Editor is a graphical tool in the Xilinx Development System that allows you to enter timing and pin location constraints. We will take advantage of this tool to not only view the constraints specified currently in the watch.ucf file, but to also add in some timing constraints of our own.

To continue with the tutorial, select the following from within Design Manager.

`Utilities → Constraints Editor`

You will be prompted to run the Translate step before launching this Utility. The following step covers the steps needed to effectively translate your design.

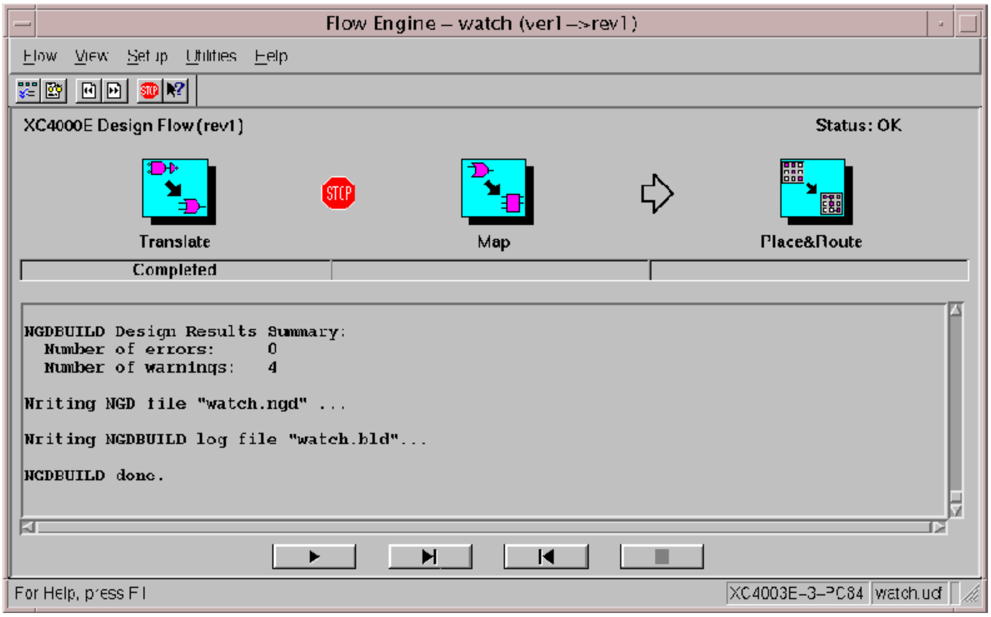
Step 3: Translating the Design

The Design Manager manages the files created during the implementation process while the Flow Engine controls the implementation process itself. The programs run by the Flow Engine use the settings supplied by the user in the options dialog box. The Flow Engine gives you complete control over how a design is processed. Typically, you should set all your options first, and then run through the entire flow by selecting “Implement” from the Design Menu.

In this tutorial, you are attempting to further define the design by setting constraints after having defined options. As stated previously, in order to invoke the Constraints Editor, you must first run the Translate step.

Select **OK** to continue with the flow.

The Flow Engine is invoked for the first time. The steps in the design flow are graphically represented in the upper half of the Flow Engine window. The status of each stage is also shown. Refer to the following figure.


Flow Engine – watch (ver1 –>rev1)
Flow View Setup Utilities Help
XC4000E Design Flow (rev1)
Status: OK
Translate
Map
Place&Route
Completed
For Help, press F1
XC4003E-3-PC84
watch.ucf

```
NGDBUILD Design Results Summary:
  Number of errors:     0
  Number of warnings:   4

Writing NGD file "watch.ngd" ...

Writing NGDBUILD log file "watch.bld"...

NGDBUILD done.
```

Figure 2-9 Translating Design

Notice the “STOP” sign placed between the Translate and MAP steps. This breakpoint has been automatically set in this situation to instruct the Flow Engine to stop after the Translate step is complete.

During translation, the program NGDBuild is executed, and performs the following functions.

- Converts input design netlists and writes results to a single merged NGD netlist. The merged netlist describes the logic in the design as well as any location and timing constraints.
- Performs timing specification and logical design rule checks
- Adds the User Constraints File (UCF) to the merged netlist

Once complete, the Flow Engine shuts down and the Constraints Editor is invoked.

Step 4: Using the Constraints Editor

The Constraints Editor is a utility that allow you to edit constraints previously defined (through a UCF file), as well as add new constraints to your design. Input files to the Constraints Editor include the following.

- NGD (Native Generic Database) file. This file serves as input to the mapper, which then outputs the physical design database, an NCD (Native Circuit Description) file.
- Corresponding UCF (User Constraint File).

By default, when the NGD file is opened, an existing UCF file with the same base name as the NGD file is used. Alternatively, you can specify the name of the UCF file.

Upon successful completion, the Constraints Editor writes out a valid UCF file. NGDBuild uses the UCF file, along with design source netlists to produce a newer NGD file that incorporates the changes made. The NGD is then read by the MAP program (the next step in the design flow). In our design, the watch.ngd file and watch.ucf file are automatically read into the Constraints Editor.

The Global Tab appears in the foreground of the Constraints Editor window. This window automatically displays all the clock nets in your design, and allows you to define the associated period, pad to setup, and / or clock to pad values.

1. Select the Period cell on the row associated with the clock net oscout. Double-click your left mouse button. This invokes the Clock Period dialog box.

   Within the Clock Signal Definition, keep the default (Specific Time) selected to define an explicit period for the clock rather designate a period which is relative to another timing specification.

2. Enter a value of 20 in the Time text box. Verify that ns is selected from the Units pull-down list. Click OK.

   Notice that the period cell is updated with the global clock period constraint we just defined (with a default 50% duty cycle)

Note: For the purpose of this tutorial, we invoked a secondary dialog by double-clicking on a cell to specify our constraint values. A new feature to the Constraints Editor in 2.1i allows for the direct entry of constraints into cells by simply clicking once.

3. Select the Ports tab from the Constraints Editor's main window.

   The left hand side displays a listing of all the current ports as defined the user. Notice that certain cells in the Location column are pre-populated with device pins locking down ports to actual pins on the target device. This information was obtained by the Constraints Editor by way of the watch.ucf file it read in.

4. Select **`File`** → **`Save`**.

   The change made within the Constraints Editor is now saved into the watch.ucf file in your current revision directory.

5. You will prompted with a reminder to rerun the Translate step. Click **`OK`**.

6. Select **`File`** → **`Exit`**

Note: Make sure you read the following procedure before starting to Map your design.

How to Stop the Design Processing Flow

Before we continue implementing our design in the Flow Engine, review the following procedure for stopping the processing of the design after the MAP step.

Warning: Because the steps for the tutorial design can often finish quickly, you should be familiar with this procedure before you start the Flow Engine.

Setting a break point anywhere in the design process is useful when you want to stop and evaluate your performance before going forward. For example, setting a breakpoint after the Translate step is useful when you want to perform a functional simulation of a design and copy the resulting *design*.ngd file to your working directory. After copying the *design*.ngd file, you can run the appropriate NGD2XXX program on the file to create functional simulation data. For more information on the NGD2XXX programs, see the appropriate chapter in the *Development System Reference Guide*.

Note: This procedure can be utilized at any time in the Flow Engine to stop after any of the steps in the design flow.

1. In this tutorial, you want to stop processing the design after the MAP step. To do this, you must set a break point to stop the Flow Engine. To stop after the MAP step from within the Flow Engine, click the stop sign toolbar icon while MAP is running.
2. The Stop After dialog box is displayed with the default setting of Configure as shown in the following figure. The list box displays the break points appropriate for the current state of the design. Because the design has not completed processing at this point, all possible break points are listed.

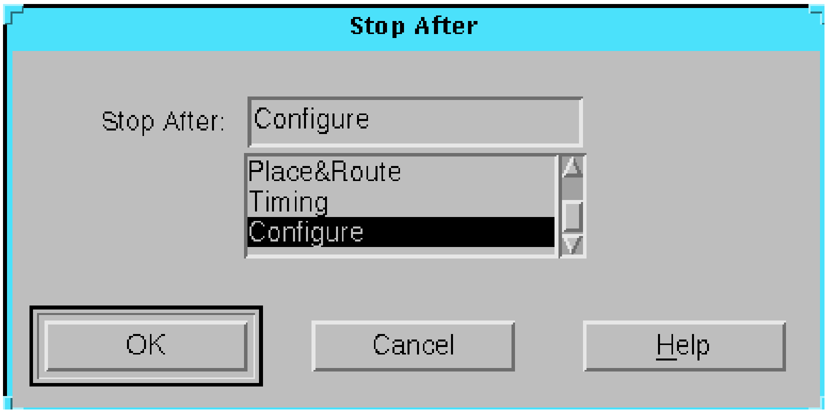
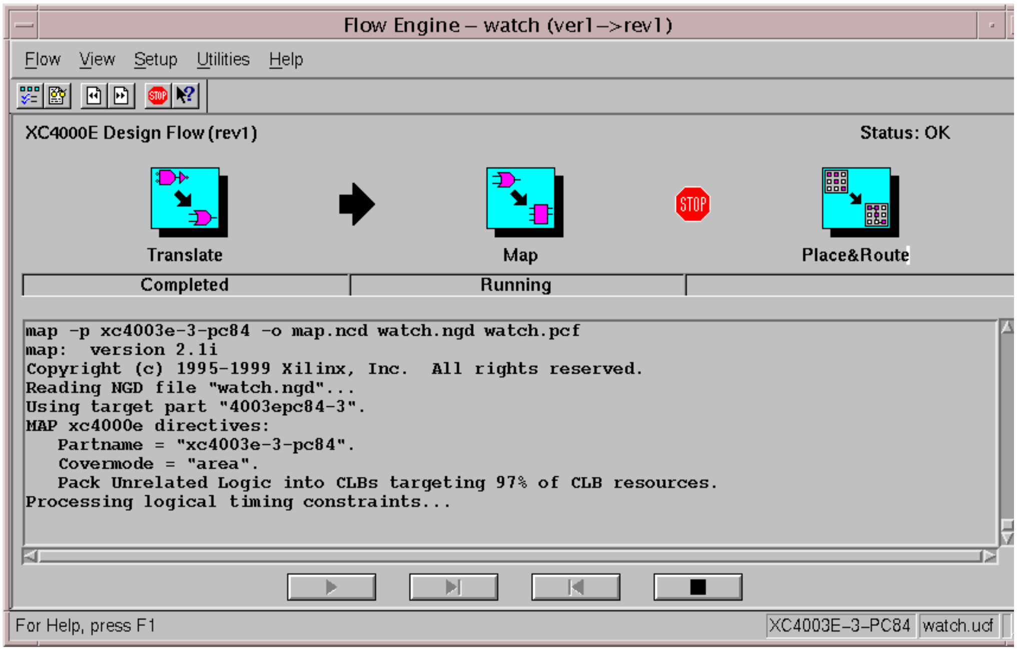


Figure 2-10 Stop After Dialog Box

3. Select MAP in the list box and click **OK**. The stop sign is added to the design flow between the Map and Place and Route steps as shown in the following figure.

Note: The status bar at the bottom of the Flow Engine window will be updated with the specified user constraints file (watch.ucf).



```
map -p xc4003e-3-pc84 -o map.ncd watch.ngd watch.pcf
map:  version 2.1i
Copyright (c) 1995-1999 Xilinx, Inc.  All rights reserved.
Reading NGD file "watch.ngd"...
Using target part "4003epc84-3".
MAP xc4000e directives:
   Partname = "xc4003e-3-pc84".
   Covermode = "area".
   Pack Unrelated Logic into CLBs targeting 97% of CLB resources.
Processing logical timing constraints...
```

Figure 2-11 Mapping Design

Starting the Flow Engine and Translating/Mapping your Design

Now that all implementation strategies have been defined (options and constraints), let's continue with the implementation of our design.

1. Select **`Design`** → **`Implement`** from the Design Manager.
2. The Flow Engine automatically detects that changes were made to your constraints file, which requires the Translate step to be re-run. In order for the changes we just made to the Constraints Editor to take affect, select YES.
3. Perform the procedure previously described in the "How to Stop the Design Processing Flow" section to stop the processing of the design.

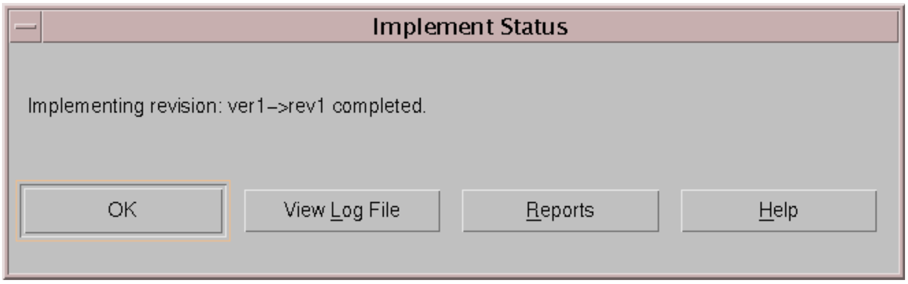
Step 5: Mapping the Design

At this point, the input netlist is being translated (once again), and merged into a single design file. Furthermore, the design will be mapped into CLBs and IOBs. After mapping, the design will be placed and routed. The final step in the design flow is the Configure step in which a configuration bitstream is created for downloading to a target device or for formatting into a PROM programming file.

Map performs the following functions:

- Allocates CLB and IOB resources for all basic logic elements in the design
- Processes all location and timing constraints, performs target device optimizations, and runs a design rule check on the resulting mapped netlist.

After the MAP step is done, the Flow Engine shuts down and the Implement Status dialog box appears, as shown in the following figure.



Implementing revision: ver1–>rev1 completed.

Figure 2-12 Implement Status Dialog Box

The following steps show you how to use the report browser.

1. Select **`Reports`** to invoke the Report Browser window. The Translation Report appears as the first report generated. The Map Report and Logic Level Timing Report files are created as a result of the Map stage completing. New reports that have not been read are denoted with a gold star in the upper left corner of the file icon, as shown in the following figure.

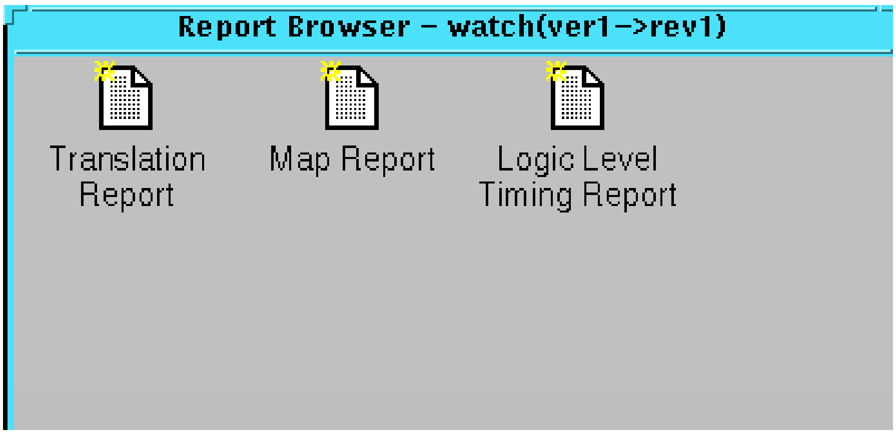


Figure 2-13 Report Browser after Running Map

2. Double-click on a report to review its output. The following table lists the types of reports and describes their contents.

   The following table lists the types of reports available to you.

Table 2-2 Report Browser Reports

Report	Description
Translation Report	Includes warning and error messages from the translation process.
Map Report	Includes information on how the target device resources are allocated, references to trimmed logic, and device utilization. For detailed information on the Map report, refer to the *Development System Reference Guide*.
Logic Level Timing Report	Provides a summary analysis of your timing constraints based on block delays and estimates of route delays. This report is produced after Map and prior to PAR (Place And Route).

3. Select **OK** to close the Implement Status dialog. Keep the Report Browser open for now. We will be evaluating some of these reports in further detail in the next section.

Notice that the Design Manager project view displays the status of the revision as (Mapped, OK). “Mapped” refers to the state of the design and is updated throughout the tutorial as the different compilation stages are completed. “OK” refers to the status of the current state and indicates no errors in the design processing.

The design has now been mapped to the target architecture. The next step involves checking the design paths for block delays.

Step 6: Using Timing Analysis to Evaluate Block Delays After Mapping

After the design is mapped, you can use the Logic Level Timing Report to evaluate the logical paths in the design. Because the design is not placed and routed yet, actual routing delay information is not yet available. The timing report describes the logical block delays and estimated routing delays. The net delays that are provided are based on an optimal distance between blocks (also referred to as *unplaced floors*).

Estimating Timing Goals With 50/50 Rule

You can get a preliminary idea of how realistic your timing goals are by evaluating a design after the map stage. A rough guideline (known as the 50/50 rule) specifies that the block delays in any single path make up approximately 50% of the total path delay after the design is routed. For example, a path with 10ns of block delay should meet a 20ns timing constraint after it is placed and routed. If your design is extremely dense, or if you are using an architecture with fewer routing resources (for example, a 4025E device versus a 4028XL), your net delays can be more than 50% of the total path delay.

Report Paths In Timing Constraints Option

Because timing constraints were defined for this tutorial design, the Report Paths in Timing Constraints option was selected. This option forces the Logic Level Timing Report to provide a period and path analysis on the constraints specified. Taking a look at the report, the period timing constraint is listed on top, as is the minimum period obtained by the tools after mapping. Because we limited our report to one path per timing constraint, we see a breakdown of a single path that contains 4 levels of logic. Notice the percentage of block (logic) delay versus routing delay for this calculation. The unplaced floors listed are estimates (indicated by the letter “e” next to the net delay) based on optimal placement of blocks.

If you do not generate a Logical Level Timing Report, PAR still processes a design based on the relationship between the block delays, floors, and timing specifications for the design. For example, if a PERIOD constraint of 8 ns is specified for a path, and there are block delays of 7 ns and unplaced floor net delays of 3 ns, PAR stops and generates an error message. In this example, PAR fails because it determines that the total delay (10 ns) is greater than the constraint placed on the design (8 ns).

Use the Logic Level Timing Report to determine timing violations that may occur prior to running PAR.

Step 7: Placing and Routing the Design

After the mapped design is evaluated to verify that block delays are reasonable given the design specifications, the design can be placed and routed. The Flow Engine can perform the following place and route algorithms.

- Timing Driven — run PAR with timing constraints specified from within the input netlist or from a constraints file
- Non-Timing Driven — run PAR and ignore all timing constraints

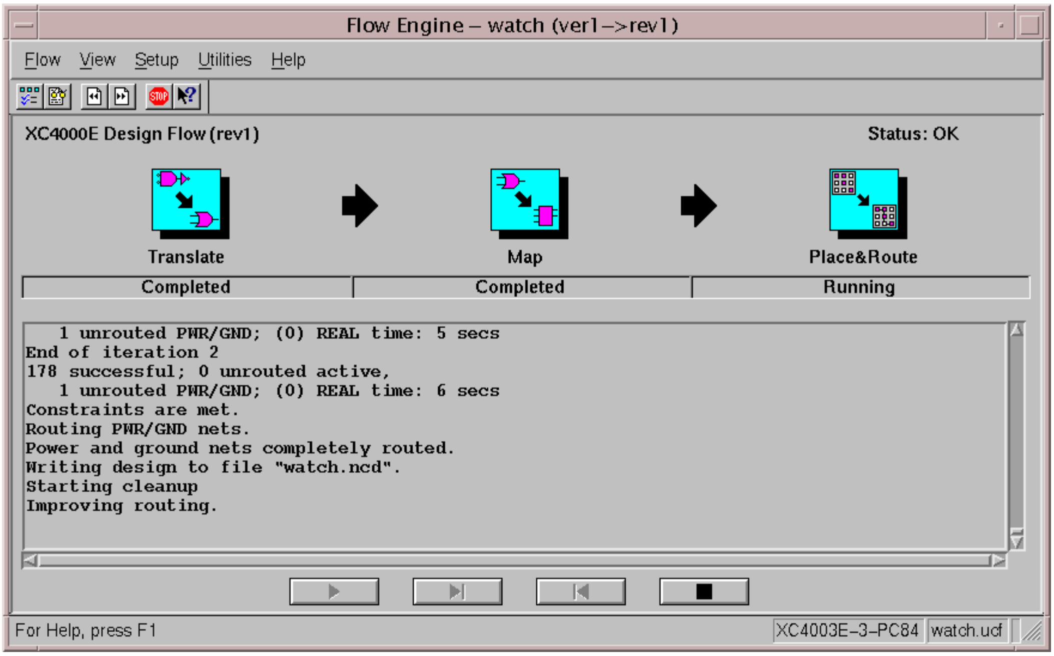
In this tutorial, timing driven placement and timing driven routing are automatically performed by PAR because timing constraints are specified for this design.

Close the Report Browser and any open reports.

To place and route the design, perform the following procedure.

1. In the Design Manager window, select **`Design`** → **`Implement`** to continue running the implementation flow.

   The Flow Engine will once again be invoked. The Status:OK message in the upper right corner indicates that no errors are generated by PAR at this point. Refer to the following figure:



```
    1 unrouted PWR/GND; (0) REAL time: 5 secs
End of iteration 2
178 successful; 0 unrouted active,
    1 unrouted PWR/GND; (0) REAL time: 6 secs
Constraints are met.
Routing PWR/GND nets.
Power and ground nets completely routed.
Writing design to file "watch.ncd".
Starting cleanup
Improving routing.
```

Figure 2-14 Placing and Routing Design

2. Review the reports generated to make sure the place and route process finished as expected.

   The four new reports created in the Report Browser are the Place and Route Report, the Pad Report, the Asynchronous Delay Report, and the Post-Layout Timing Report, as shown in the following figure and described in the following table.

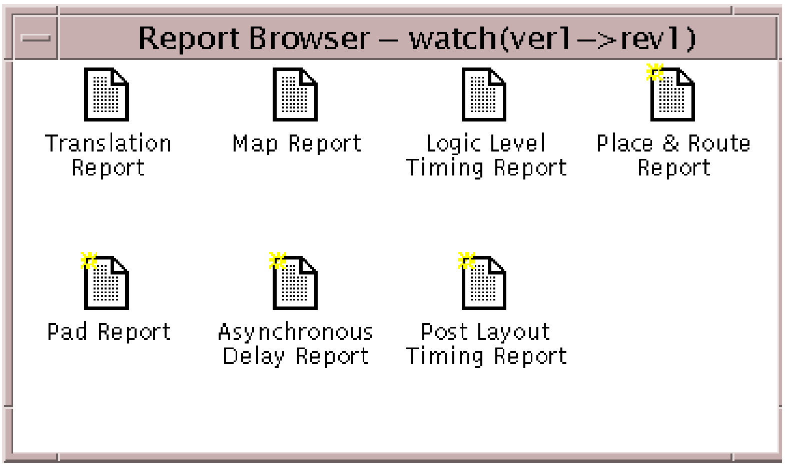

Report Browser – watch(ver1->rev1)
Translation Report
Map Report
Logic Level Timing Report
Place & Route Report
Pad Report
Asynchronous Delay Report
Post Layout Timing Report

Figure 2-15 Reports Available After Place & Route

Table 2-3 Description of Reports Available After Place & Route

Report	Description
Place & Route Report	Provides a device utilization and delay summary. Use this report to verify that the design successfully routed and that all timing constraints were met.
Pad Report	Contains a report of the location of the device pins. Use this report to verify that pins locked down were placed in the correct location.
Asynchronous Delay Report	Lists all nets in the design and the delays of all loads on the net.
Post-Layout Timing Report	Incorporates both the logic and routing delays to generate an evaluation of the design's timing constraints, clock frequencies, and path delays.

Note: In the Design Manager window, the status of the current version / revision is now (Routed, OK).

Step 8: Evaluating Post-Layout Timing

After the design is placed and routed, a Post Layout Timing Report is generated by default to verify that the design meets your specified timing goals. This report evaluates the logical block delays and the routing delays. The net delays are now reported as actual routing delays after the place and route process (indicated by the letter “R” next to the net delay).

Double-click on the Post Layout Timing Report to open it. Following is a summary of this report.

- The minimum period value increased due to the actual routing delays.
- After the Map step, logic delay contributed to about 80% of the minimum period attained. The post-layout report indicates that the logical delay value decreased somewhat. The total unplaced floors estimate changed as well. Routing delay after PAR now equals about 31% of the period; a true report of net delays after the place and route step.
- The post-layout result does not necessarily follow the 50/50 rule previously described because the worst case path includes primarily component delays. After the design was mapped, block delays constituted about 80% of the period. After place and route, the majority of the worst case path is still made up of logic delay. Since total routing delay makes up only a small percentage of the total path delay, spread out across three nets, expecting this to be reduced any further is unrealistic. In general, you can reduce excessive block delays and improve design performance by decreasing the number of logic levels in the design.

Step 9: Creating Timing Simulation Data

After your design is placed and routed and the timing is statically verified, the next step is to create timing simulation data. To create timing simulation data, perform the following steps in the Design Manager.

1. Select **`Design`** → **`Options`** to open the Options dialog box.
2. Select the simulator that corresponds to your design entry tool from the Simulation drop-down list in the Program Options section of the dialog box.

3. Click **`OK`** to close the Options dialog box.
4. Select **`Design`** → **`Implement`** from the Design Manager

   Within the Flow Engine, you will now notice a new stage appear directly after Place & Route. This new stage, called Timing(Sim), is solely dedicated to producing timing simulation data. In the tutorial, this stage did not appear originally because the Program Option for Simulation was not selected to a specific simulator in the initial pass. By default, this option is set to OFF. For all designs, you have the choice of selecting all options at the beginning of the design processing, or coming back to set them later.

   During the Timing(Sim) step, the Flow Engine runs the NGDAnno program to create a back-annotated NGD file. The NGD file is then used as input to one of the NGD2XXX programs to produce the preferred simulation file format. By default, the files created are named *time_sim*. To make it easy to find the output files for your third-party simulation environment, the files are automatically copied to your working directory.

Step 10: Creating Configuration Data

The next step is creating configuration data. This step includes creating a bitstream for the target device by running the configure step, as follows:

1. Select **`Design`** → **`Options`** to open the Options dialog box.
2. Select **`Default`** from the drop-down list for Configuration Program Options.
3. Click the **`Edit Options`** button corresponding to Configuration, which just became enabled with our selection of Default.

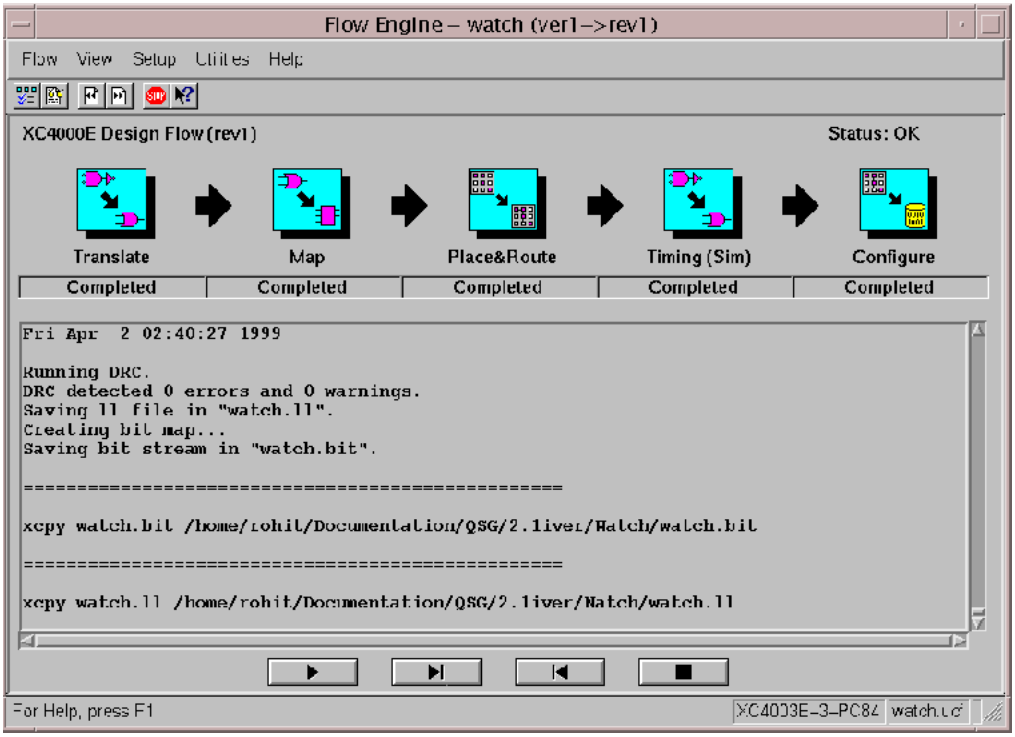
   The XC4000 Configuration Options dialog box appears. The configuration templates set options that define the initial configuration parameters, start-up sequence, readback capabilities, and other advanced features. In this tutorial, a configuration file is created that can be used for programming, verifying, and debugging XC4000E designs.
4. In the Configuration tab, verify that **`PullUp`** is selected next to the Done pin, and that the **`Perform CRC During Configuration`** option is selected.

5. Select the Readback tab, and verify that **`CCLK`** is selected as the readback clock.
6. Click **`OK`** to close the XC4000 Configuration Options dialog box.
7. Click **`OK`** to close the Options dialog box.
8. Select **`Design`** → **`Implement`** from the Design Manager.

   The Flow Engine comes up, running the BitGen program in the newly added Configure stage. BitGen creates the *design_name*.bit and *design_name*.ll files (in this tutorial, the watch.bit and watch.ll files). The *design_name*.bit file is the actual configuration data. The *design_name*.ll file is the logical allocation file that is used during hardware debugging to determine the location of the probable points in the design. These files are automatically copied to your working directory. Verify that they are in your working directory.

   For more information on device readback, please refer to the latest version of the Watch Design Hardware Verification Tutorial, located at http://support.xilinx.com/support/techsup/tutorials/index.htm.

   The following figure shows the Flow Engine window after the configure step is finished.



```
Fri Apr  2 02:40:27 1999

Running DRC.
DRC detected 0 errors and 0 warnings.
Saving ll file in "watch.ll".
Creating bit map...
Saving bit stream in "watch.bit".

==================================================

xcpy watch.bit /home/rohit/Documentation/QSG/2.1iver/Watch/watch.bit

==================================================

xcpy watch.ll /home/rohit/Documentation/QSG/2.1iver/Watch/watch.ll
```

Figure 2-16 Configuring Design

9. The Flow Engine saves the configuration options in the BitGen Report. Review the report using the Report Browser. Verify that the specified options were used when creating the configuration data.

Step 11: Using the PROM File Formatter

If you are going to program a single device using the Hardware Debugger, all you need is a *design*.bit file. If you are going to program several devices in a daisy chain configuration, or program your devices using a PROM, you must use the PROM File Formatter (PFF) to create a PROM file. The PROM File Formatter accepts any number of bitstreams and creates one or more PROM files containing one or more daisy chain configurations.

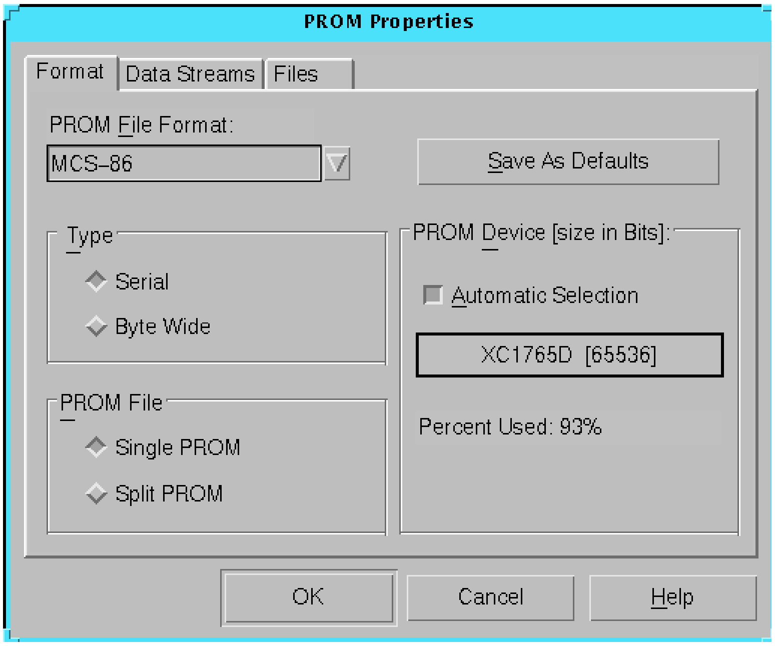
1. To start the PROM File Formatter, click the PROM File Formatter icon in the toolbox in the Design Manager.

   The PFF starts with a default PROM that matches the currently selected (configured) revision. At this point, you can add additional bitstreams to the daisy chain; create additional daisy chains; remove the current bitstream and start over; or immediately save the current PROM file configuration.

   The status bar at the bottom of the PFF window displays the PROM format, data format, current PROM size, and percentage of the selected PROM used by the current PROM configuration. The currently selected PROM is an XC1765D. 53,984 bits of data are required to hold the configuration bitstream for the XC4003E target device for this tutorial. The PFF determined that an XC1765D is the correct PROM because it can hold up to 65,536 configuration bits (or 82% full).

   The right half of the PFF window is a directory structure used for locating bitstreams. Only files with a .BIT extension are shown in the list. For detailed information on using the PROM File Formatter to create daisy chains or complex PROM configurations, see the *PROM File Formatter Guide*. This tutorial describes how to save the default PROM file.

2. Select **`File`** $\rightarrow$ **`PROM Properties`** to open the PROM Properties dialog box, shown in the following figure.


PROM Properties

Format | Data Streams | Files

PROM File Format: MCS-86

Type: [✓ Serial] Byte Wide

PROM File: [✓ Single PROM] Split PROM

PROM Device [size in Bits]: Automatic Selection

XC1765D [65536]

Percent Used: 93%

Figure 2-17 PROM Properties Dialog Box with Single PROM

3. Select the following options in this dialog box.
   - PROM File Format from the drop-down list
   - PROM Type
   - Number of PROMS used to hold the data

   If you have more data than space available in the PROM, you must split the data into several individual PROMs with the Split PROM option. In this case, only a single PROM is needed. Click **OK** to accept the PROM Properties.

4. Select **`File`** → **Save** to save the PROM file.

5. Specify your working directory as the area where the PROM Description File will be saved.

   The PROM File Formatter saves both the PROM file (watch.mcs) and a PROM Description File (watch.pdr). The PDR file can be re-opened if any changes are required. Verify that the files exist in your directory.

6. Select **`File`** → **`Exit`** to close the PROM File Formatter.

   This completes the tutorial. For more information on the Alliance design flow and implementation methodologies (especially some of the tools and programs that were not covered as part of this tutorial), please reference the online version of the Software Manuals at http://support.xilinx.com

Chapter 3

Using the Software

This chapter provides an overview of the Xilinx Development System. The standard flow from netlist to PROM file is described, including information on options, reports, simulation netlists, constraints, floorplanning, and guided implementations. Advanced flows, such as re-entrant routing and multi-pass place and route, are also described. This chapter includes the following sections.

- “Using the Xilinx Tools”
- “Xilinx Design Flow”
- “Selecting Options”
- “Using Design Constraints”
- “Guiding a Design with Floorplanner Files”
- “Static Timing Analysis”
- “Creating Simulation Files”
- “Downloading a Design”
- “Multi-Pass Place and Route”
- “Guiding an Implementation”

Note: For the latest information regarding the Design Manager tools and functions, see the “Legacy Information” Appendix of the *Design Manager/Flow Engine Guide.*

Using the Xilinx Tools

To start the Xilinx tools double click on the Design Manager icon, or enter the following at the command line to start the Design Manager.

```
xilinx
```

You can also start the Design Manager by entering the following at the command line.

dsgnmgr

Xilinx Design Flow

The "Xilinx Design Flow" figure shows the processing steps and flow of files in and out of the Design Manager. The "Detailed Design Flow" figure is a more detailed look at the various programs invoked during the design implementation process.

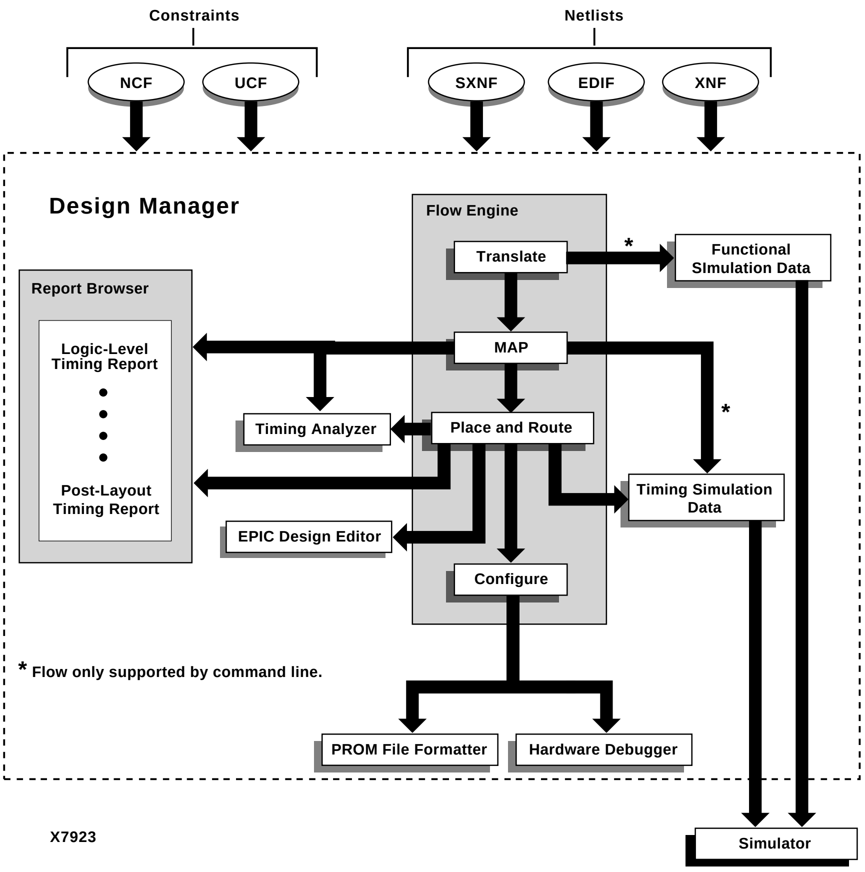


Figure 3-1 Xilinx Design Flow

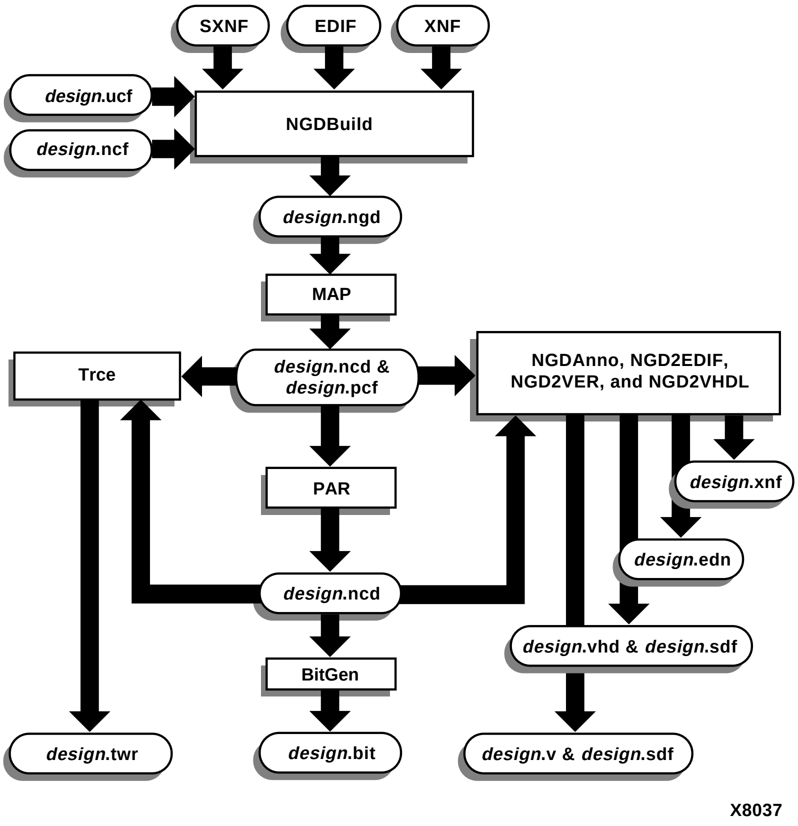


Figure 3-2 Detailed Design Flow

Using the Design Manager

The following section provides basic information about using the Design Manager tool.

Note: Refer to the *Design Manager/Flow Engine Guide* for detailed information on using the Design Manager.

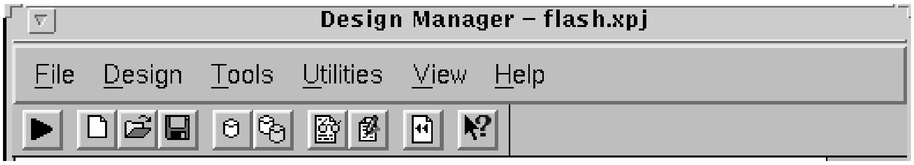


Figure 3-3 Design Manager Menu

Creating a Project

Use the following steps to create a new project in the Design Manager.

1. Select **`File`** → **`New Project`** from the Design Manager menu or click the New Project toolbar button. The New Project dialog box appears.
2. Specify a design file to open with one of the following methods.
   - In the Input Design field, type the name of a design file to open.
   - Click the Input Design **`Browse`** button to the right of the Input Design box to select the top level input netlist. Click **`OK`**.

Note: The Design Manager automatically creates a subdirectory named xproj under the input design directory and uses it as the work directory. The Design Manager uses the xproj subdirectory to store all the data files for the project. If you want to change this default work directory, type a path in the Work Directory field or use Browse to select a directory.

3. In the New Project dialog box, click **`OK`**.

   After your design is loaded, the Design Manager window appears, configured for the loaded design.

For information on using the Xilinx-supplied interface tools for Synopsys, Viewlogic, Mentor Graphics, or Cadence designs, see the appropriate appendix of this manual, or refer to the Interface User Guide for your respective tool.

Implementing Your Design

1. Select **`Design`** → **`Implement`** from the Design Manager menu or click the Implement toolbar button. The Implement dialog box appears.
2. Select the part and click Run. The Design Manager automatically creates a new version and revision. Additional versions are created when the netlist is modified and re-implemented. Additional revisions are created when the same netlist is re-implemented with new options or constraints. The Design Manager invokes the Flow Engine to process your design.

Using the Flow Engine

The Flow Engine allows you to process and control the implementation of your design, as well as guide your implementation revisions. The following figure shows the various steps followed by the Flow Engine to process your designs.

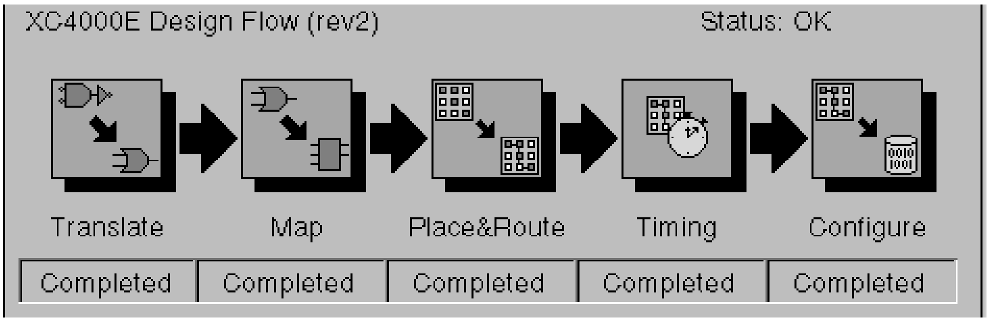


Figure 3-4 Flow Engine Design Steps

Translating Your Design

The Flow Engine's first step, Translate, merges all of the input netlists by running the NGDBuild program.

Mapping Your Design

Mapping your design is the next step in the design flow. Map optimizes the gates and trims unused logic in the merged NGD netlist.

Map also maps your design's logic resources and performs a physical design rule check.

Placing and Routing Your Design

After mapping, the Flow Engine places and routes your design. The PAR (Place and Route) program is invoked to optimally place and route the mapped CLBs and IOBs in your design. If there are timing constraints on any of the logic components, PAR attempts to minimize those delays by moving the corresponding logic blocks closer together. In the route stage, the logic blocks are assigned specific interconnect elements on the die. PAR attempts to minimize any delays by selecting a faster interconnect.

Configuring Your Design

After placing and routing your design, the Flow Engine translates the physical implementation into a binary stream that is used to program an FPGA.This binary stream is saved as a configuration file (.bit) using the BitGen program.

Analyzing Reports with the Design Manager

Design Manager reports provide information on logic trimming, logic optimization, timing constraint performance, and I/O pin assignment. To access the reports, select the following from the Design Manager menu.

`Utilities` → **`Report Browser`**

To open a specific report, double click on its icon, as shown in the following figure.

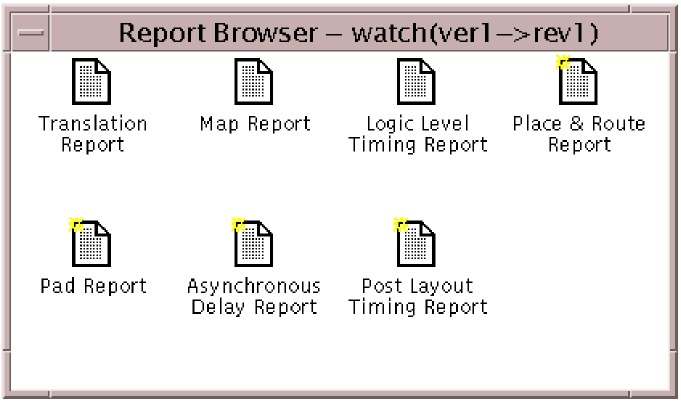

Report Browser – watch(ver1->rev1)
Translation Report
Map Report
Logic Level Timing Report
Place & Route Report
Pad Report
Asynchronous Delay Report
Post Layout Timing Report

Figure 3-5 Report Browser

Translation Report

The Translation Report contains warning and error messages from the three translation processes: conversion of the EDIF or XNF style netlist to the Xilinx NGD netlist; timing specification checks; and logical design rule checks. The report lists the following.

- Hierarchical blocks that are missing or cannot be translated
- Invalid or incomplete timing constraints
- Output contention, loadless outputs, and sourceless inputs

Map Report

The Map Report (.mrp file) contains warning and error messages detailing logic optimization and logic mapping to physical resources. The report lists the following information.

- Removed Logic — Sourceless and loadless signals can cause the removal of an entire chain of logic. Each deleted element is listed with progressive indentation so you can easily identify the origins of the removed logic sections; deletion statements are not indented.
- Added or expanded logic due to speed optimization.
- The Design Summary lists the number and percentage of used CLBs, IOBs, flip-flops, and latches. It also lists the use of architec-

ture-specific resources such as global buffers and boundary scan logic.

Place and Route Report

The Place and Route Report (.par file) contains the following information.

- Design Score — The Design Score measures the relative goodness of your design; a lower score is better. Because this score is strongly dependent on the nature of your design and the targeted part, meaningful score comparisons can only be made between iterations of the same design targeted for the same part.
- The Number of Signals Not Completely Routed should be zero for a completely implemented design. If not, you may be able to improve results by using the re-entrant route flow or the multi-pass place and route flow. See the “Advanced Implementation Flows” section at the end of this chapter.
- The timing summary at the end of the report contains the timing performance of your design. For information on timing constraint performance and synchronous delays, refer to the “Static Timing Analysis” section later in this chapter.

Pad Report

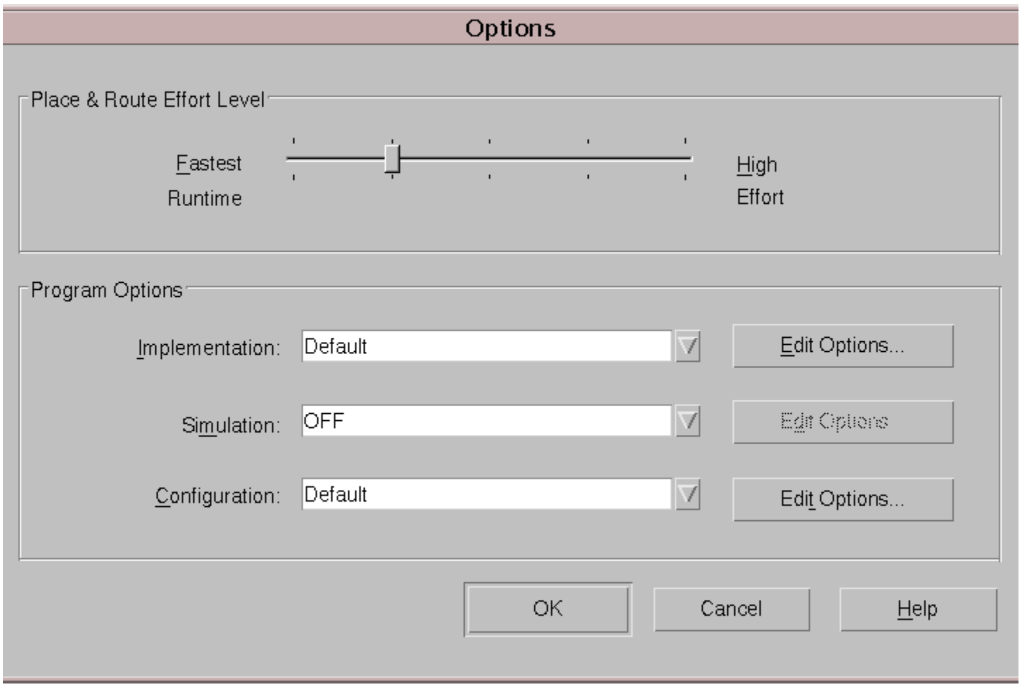
The Pad Report lists your design’s pinout sorted by signal name, and then by pin number.

Selecting Options

Options specify how your design is optimized, mapped, placed, routed, and configured. Options are grouped as implementation templates or configuration templates. Each template defines an implementation or configuration style. For example, an implementation style can be Quick Evaluation, while another can be Timing Constraint Driven.

You can have multiple templates in a project. You can use templates to select an implementation or configuration style. To access the options and templates, follow these steps.

1. Select **`Design`** → **`Implement`** from the Design Manager menu or click the Implement toolbar button. The Implement dialog box appears.
2. Select the Options button in the Implement dialog box. The Options dialog box appears as shown in the following figure.



Implementation: Default
Simulation: OFF
Configuration: Default

Figure 3-6 Options Dialog Box

3. Select the Edit Template button for Implementation or Configuration to access the associated template. The Implementation Template or Configuration Template dialog box appears. The options in this box depend on the target device family. For information on how to use the template options, see the *Design Manager/Flow Engine Guide*.

Using Design Constraints

The Xilinx tools allow you to control the implementation of your design by entering constraints. You can enter constraints during the design and implementation phases of the design flow. During the design phase, you can enter constraints as follows.

- Add constraints to your schematic design
- Add constraints to your design in your synthesis tool
- Enter constraints in the Xilinx Constraints Editor

You can apply location and timing constraints to your design. Use location constraints to control the mapping and positioning of the logic elements in the target device. The most common location constraints are pad constraints. Pad constraints are used to lock the pins of your design to specific I/O locations so that the pin placement is consistent from revision to revision. Use timing constraints to specify how fast a path must be to meet your speed requirements. You can use timing constraints for the placement and routing of your design.

Constraints entered directly in your input design are known as design constraints, and are eventually placed in your design netlist. If you want the constraints separated from your input design files, or if you want to modify your constraints without re-synthesizing your design, you can create a User Constraints File (UCF) in the Constraints Editor. This file is read by NGDBuild during the translation of your design, and is combined with an EDIF or XNF netlist into an NGD file. If a UCF file exists with the same name as the top-level netlist, it is automatically read. Otherwise, you must specify a file name for User Constraints in the Options dialog box.

Adding Constraints with the Constraints Editor

The Constraints Editor is a graphical tool in the Xilinx Development System that allows you to enter timing constraints and pin location constraints. You can enter constraints in the graphical interface without understanding UCF file syntax. The Constraints Editor passes these constraints to the implementation tools through a UCF file.

The Constraints Editor accepts the following input files.

- A valid NGD file, which is a Xilinx logical design database file. This file serves as input to the Map program, which generates the physical design database (NCD).

- A corresponding UCF (User Constraint File), which contains logical constraints.

By default, when the NGD file is opened, an existing UCF file with the same base name as the NGD file is used. Alternatively, you can specify the name of the UCF file.

The Constraints Editor writes out a valid UCF file and a valid NGD file.These files are processed by the Map program, which generates a PCF (Physical Constraints File).

Note: For more information, see the *Constraints Editor User Guide*.

Guiding a Design with Floorplanner Files

The Floorplanner tool generates an MFP file that contains mapping and placement information. You can use this file as a guide for mapping an implementation revision.

Note: If you use an MFP file as a guide file, you cannot guide mapping using the Set Guide File(s) command Custom option. Also, the Floorplanner is only available for XC4000 and Spartan devices.

To guide your design with floorplan files, follow these steps.

1. In the Design Manager project view, select an implementation revision that has been mapped and modified using the Floorplanner.

   For more information on the Floorplanner, refer to the *Floorplanner Reference/User Guide*.

2. Select **`Design`** → **`Set Floorplan File(s)`** from the Design Manager.

   The Set Floorplan File(s) dialog box appears.

3. Select a floorplan guide design from the Floorplan Design drop-down list.

   - Select an existing implementation revision.
   - Select **`None`** if you do not want to guide the design. Select **`Project Clipboard`** to guide from the implementation revision copied to your project clipboard. If no data exists in the clipboard or if you want to copy new data to the clipboard, use the Copy Floorplan Data to Project Clipboard option in the Implement dialog box.

- Select **`Custom`** to guide from any mapped file in your file system, including designs not generated from within the Design Manager. This option invokes the Custom dialog box in which you can specify your floorplan guide files. Specify an FNF file for the Floorplanning File field and an MFP file for the Floorplanned Guide File field.

4. The Flow Engine uses the selected file to guide the implementation.

Static Timing Analysis

Timing analysis can be performed at several stages in the implementation flow to gauge delays. A post-map timing report can be generated to evaluate the effects of logic delays on timing constraints, clock frequencies, and path delays. A post-place-and-route timing report, that incorporates both logic and routing delays, can be generated as a final evaluation of the design's timing constraints, clock frequencies, and path delays. Detailed timing constraint, clock, and path analysis for post-map or post-place-and-route implementations can be accomplished by using the interactive Timing Analyzer tool.

Static Timing Analysis After Map

Post-map timing reports can be very useful in evaluating timing performance. The report uses real block delays and estimates for the route delays. Although the delays are estimates, they provide valuable information.

If logic delays account for a significant portion (> 50 percent) for the total allowable delay of a path, the path may not be able to meet your timing requirements once the real routing delays are added. In fact, if the logic-only-delays exceed the total allowable delay for a path or constraint, then the place and route process need not be run since the routing delays will only cause the path's timing to degrade. Routing delays typically account for 40 percent to 60 percent of the total path delays. By identifying problem paths, you can mitigate potential problems before investing time in place and route. You can redesign the logic paths to use fewer levels of logic, tag the paths for specialized routing resources, move to a faster device, insert flip flops in the path, or allocate more time for the path.

If logic-only-delays account for much less (<15 percent) than the total allowable delay for a path or timing constraint, then very low effort levels can be used by the place and route tool. In these cases, reducing effort levels allow you to decrease run times while still meeting performance requirements.

Static Timing Analysis After Place and Route

Post-PAR timing reports incorporate real block and real route delays to provide a comprehensive timing summary. If a placed and routed design has met all of your timing constraints, then you can proceed by creating configuration data and downloading a device. If you identify problems in the timing reports, you can try fixing the problems by increasing the effort level, using re-entrant routing, or using multi-pass place and route. You can also redesign the logic paths to use less levels of logic, tag the paths for specialized routing resources, move to a faster device, insert flip flops in the path, or allocate more time for the paths.

You can identify paths that can be ignored, or identified as slower exceptions.

Edit the implementation template to modify the placer effort level. For information on re-entrant routing or multi-pass place and route, see the “Advanced Implementation Flows” section at the end of this chapter.

Summary Timing Reports

Implementing a design in the Flow Engine can automatically generate summary timing reports. The summary reports show timing constraint performance and clock performance. To create summary timing reports, use the following steps.

1. Open the Options dialog
    - For a post-MAP report, select the Produce Logic Level Timing Report button
    - For a post-PAR report, select the Produce Post Layout Timing Report button
2. To modify the reports to detail path delays or paths failing timing constraints, do the following.
    - Edit the Implementation template

- Select the Timing Reports tab
- Select a report format

3. After MAP or timing analysis is finished, the Logic Level Timing or Post Layout Timing report appears in the report browser.

Detailed Timing Analysis

To perform detailed timing analysis, select the following from the Design Manager.

`Tools` → `Timing Analyzer`

You can specify specific paths for analysis, discover paths not covered by timing constraints, and analyze the timing performance of the implementation based on another speed grade. For path analysis, use the following steps.

1. Choose sources; from the Timing Analyzer menu, select the following.

   `Path Filters` → `Path Analysis Filters` → `Select Sources`

2. Choose destinations; from the Timing Analyzer menu, select the following.

   `Path Filters` → `Path Analysis Filters` → `Select Destinations`

3. To create a report, select the following.

   `Analyze` → `All Paths`

4. To switch speed grades, select the following.

   `Select Options` → `Speed Grade`

After a new speed grade is selected, all new Timing Analyzer reports will be based on the design running with new speed grade delays. The design does not have to be re-implemented, because the new delays are read from a data file.

Creating Simulation Files

Once the design is implemented, a timing simulation can be performed to test the timing requirements and functionality of your design. Timing simulation can save considerable time by reducing time spent debugging test boards in the lab. Functional simulation can help you to further save time by uncovering design bugs before running Place and Route.

The Xilinx tools allow you to create simulation data after each major processing step. This means that you can create functional simulation netlists after the design has been merged together by NGDBuild in the Translate process, and timing simulation netlists after the design has been placed and routed by PAR. Additionally, you can create simulation data after the design has been mapped, or after the design has been placed but not routed.

Simulation data created after the design has only been mapped contains timing data based on the CLB and IOB block delays, and most net delays are zero.

Post-MAP simulation allows you to ensure that the design's current implementation will give the place and route software sufficient margin to route the design within your timing requirements.

Simulation data created after the design has been placed but not routed, contains accurate block delays and estimates for the net delays. Post-place simulation can be used as an incremental simulation step between post-MAP simulation and a complete post-route timing simulation.

Creating Timing Simulation Data

Follow these steps to create timing simulation data.

1. Select **`Design`** → **`Implement`** from the Design Manager menu or click the Implement toolbar button. The Implement dialog box appears.
2. Select the Options button in the Implement dialog box. The Options dialog box appears.
3. Select the Produce Timing Simulation Data option.
4. In the same dialog box, click on the Edit Template button for simulation. Select the interface tab in the Simulation Template dialog box.
5. On the General tab, select one of the simulation netlist formats (EDIF, VHDL, or Verilog®). If you selected EDIF, go to the EDIF

tab, and select a CAE Vendor. If you select VHDL or Verilog, go to the VHDL / Verilog tab and select the options you want to use for simulation.

6. On the General tab, select the Correlate Simulation Data to Input Design option if you are using a simulation stimulus file or test fixture that was used for functional simulation, and contains signal names that were optimized out of your design during implementation.

With these options selected, the Flow Engine automatically creates a post-route simulation netlist in the selected format during the timing stage. To access the simulation netlist in the Design Manager, perform the following steps.

1. Select your project revision.
2. Select **`Design`** → **`Export`**. In the Export dialog box, select Timing Simulation Data and enter the export directory for the file.
3. Select **`OK`**. The listed netlist is copied to the selected directory. Use the netlist as input to your simulator to perform a timing simulation.

Note: For more information, see the *Development System Reference Guide.*

Creating Functional Simulation Data

Functional simulation netlists should be created using tools from your simulation vendor interface and the Alliance software tools. The implementation processes do not need to be invoked to create functional simulation netlists. However, if your design contains modules with varying netlist formats that the Xilinx interface software is unable to process, you can run NGDBuild on the design to create a single design_name.ngd and then create a simulation netlist using a translation tool: NGD2VHDL, NGD2VER, or NGD2EDIF. The following commands create a functional simulation netlist.

`ngdbuild` *design_name*

`ngd2edif` *design_name*

Downloading a Design

An implemented design can be downloaded directly from your PC or workstation, using the Hardware Debugger program and the XChecker cable, the Parallel Cable III, or the MultiLINX cable.

The Hardware Debugger can download a bit file or a PROM file.

For more information on downloading, see the *Hardware Debugger User Guide* and the *Hardware User Guide.*

Creating a PROM

An FPGA or daisy chain of FPGAs can be configured from serial or parallel PROMs. The PROM File Formatter can create MCS, EXO, or TEK style files. The files are read by a PROM programmer that turns the image into a PROM.

A HEX file can also be used to configure an FPGA or a daisy chain of FPGAs through a microprocessor. The file is stored as a data structure in the microprocessor boot-up code.

In-Circuit Debugging

Once a design has been downloaded to an FPGA, snapshots of internal signal states can be captured and read using the Hardware Debugger program and the XChecker cable. You can display the signal states as waveforms in the Hardware Debugger. This capability allows you to test and debug your design in a real-time environment as it interfaces with components on your board. You can also control the states of your state machines, by controlling when clock edges are sent to your system clock input.

For more information on in-circuit debugging, the Hardware Debugger, or the XChecker cable, see the *Hardware Debugger Guide.*

Advanced Implementation Flows

The place and route software, PAR, has features that allow it to process complex designs that have tight timing requirements and/or are difficult to route. PAR options can be varied in many different ways. This section shows the most common strategies.

Re-Entrant Route

PAR can take an implemented design as an input, and use it as the starting point for routing. If your design is placed but not routed, PAR will use the placement and just spend time routing the design. If your design is partially routed, PAR will use the existing placement and routing and only spend time routing the unrouted signals. If your design is completely placed and routed but not meeting timing specifications, PAR can start from where it left off and continue re-routing the design to come up with an implementation that meets your timing specifications.

As PAR is running, it continually updates the NCD file with its current placement and routing information. PAR can use a placed NCD file for re-entrant routing. To perform re-entrant routing, follow these steps.

1. In the Design Manager, select the implemented revision, and select the Flow Engine button in the toolbox.
2. In the Flow Engine, select the following.

   Setup → FPGA Re-entrant
3. In the Setup Re-entrant Route dialog box, select the Allow Re-Entrant Routing button, which enables the re-entrant route options.
4. If meeting timing specifications is a critical goal for the route, select the Use the Timespecs button during re-entrant route. If meeting timing specifications is not critical, deselect the button because timing driven route takes longer than non-timing driven route.
5. Select the number of re-entrant routing passes. If Auto is selected, PAR performs routing iterations until it stops making significant progress or until your design constraints have been fully met.
6. Select the number and type of cleanup passes. Cleanup passes are run after the initial routing passes are complete. The effectiveness of the type of cleanup passes depends on the design, device, and constraints of the implementation. The best methodology is to select no more than three passes for each (in most cases, a single pass for each is sufficient), and use the PAR report to determine which is most effective. Then try using more cleanup passes of that style.

7. After you have selected your options, click **OK**. The Place and Route icon in the Flow Engine displays a loop back arrow and the Re-Entrant route label.

If you are specifying timing or location constraints, you may want to relax them to give PAR more flexibility. If you modify the UCF file, you must step the Flow Engine back and run Translation in order to incorporate the changes. Since your design is already implemented, step back to the beginning of Place & Route using the Step Backward button at the bottom of the Flow Engine, and then click the button to start again.

Multi-Pass Place and Route

If a design has not completed routing or the meeting of timing constraints, then you can use PAR to perform a more extensive search for a solution. PAR can produce multiple placed and routed revisions, each revision with varying implementations. PAR scores each implementation, choosing the best revisions based on the score. By choosing the best implementation from a large population, PAR is more likely to find a solution that meets your requirements.

If you are using the Xilinx software on networked UNIX workstations, you can significantly reduce run time by running the place and route passes in parallel on separate machines. To execute Multi-Pass Place and Route, perform the following steps.

1. In the Design Manager, be sure to select a version and not a revision, and then from the menu select the following.

   Design → FPGA Multi-Pass Place and Route

2. In the FPGA Multi-Pass Place and Route dialog, select a value for the Initial Placement Seed (Cost Table). The Initial Placement Seed is a value that initializes the Place and Route algorithms. Each iteration receives an incremented value of the starting strategy. For initial runs, set the Seed to 2, since 1 was used in your previous single- pass run.
3. Select the Place and Route passes to execute.
4. Select the number of iterations to save. Based on the design score, only the files from the best runs are saved. If you are using a UNIX workstation, and want to use the Turns Engine to run on multiple UNIX workstations, select a nodelist file. A nodelist file

is a user-created ASCII file that lists the names of the workstations on which you want to run. Each name should be on a separate line. There should not be any tabs or spaces.

5. Click OK to start the Multi-Pass Place and Route Process.

Guiding an Implementation

During the design process your design may be modified and implemented many times. In most cases, parts of your design do not change from one implementation to the next. Guiding your design accelerates iterative implementations by reusing the unchanged sections from a previous implementation on current implementations. This is advantageous because the software spends time generating implementations only for sections of your design that have changed. The guide process is used during map, place, and route, and can significantly reduce design run times.

The guide process is more effective when the net names and instance names in your design remain constant between iterations, except for those specific parts of your design that are modified at the source level. This is generally true for schematic-based designs, but not for synthesis-based designs. For this reason, Xilinx does not recommend using guide for most synthesis-based designs.

Specifying a Guide Design

To select a previous implementation to guide a current implementation, select the following in the Design Manager.

Design → Set Guide File(s)

The Set Guide File(s) dialog box appears. In the Guide Design field, you can select previously implemented revisions, Project Clipboard, Custom, or None.

- Project Clipboard

  The project clipboard is used to save the guide data of revisions that are overwritten. You can save guide data to the project clipboard by selecting the Copy Guide Data To Project Clipboard option in the Implement dialog box.

- Custom

Use the Custom option to guide from any mapped, routed, or fitted file in your file system, including designs not generated from within the Design Manager. In the Custom dialog box, enter a mapped NCD file in the Mapping Guide File field. Enter a placed and routed NCD file in the Guide File field.

- None

  Select the None option if you do not want to guide your design.

Exact Guide Mode

When guiding in exact mode, the unchanged logic is not modified in any way. This mode is fastest, but least flexible. Use this mode if the design iteration requires only minor changes. Exact mode is the default value. It can be selected by having the Match Guide Design Exactly button pressed in the Options dialog.

Leveraged Guide Mode

When guiding in leveraged mode, the mapping, place, or route of the unchanged logic can be modified if the tools need to make layout changes to accommodate new logic. Use this mode if significant changes have occurred in your design.

Leveraged mode is automatically selected when the Match Guide Design Exactly button is not selected in the Options dialog.

Appendix A

Alliance FPGA Express Interface Notes

This appendix provides information on installing and using the Alliance FPGA Express and the Xilinx Alliance Series release. Synopsys and the Xilinx CD-ROM documentation are referenced to help you find additional information. The Alliance FPGA Express is FPGA Express software purchased from Synopsys. Foundation Express is the FPGA Express software bundled with the current release of Foundation and is purchased from Xilinx. All references to FPGA Express in this appendix refer to Alliance FPGA Express. For more information on Foundation Express, refer to the *Foundation Series 2.1i Quick Start Guide*.

FPGA Express is a Verilog/VHDL compiler designed to work with Windows 95/98 and Windows NT v4.0. FPGA Express can process either Verilog or VHDL files. This tool writes out XNF files (EDIF for Virtex designs) which are fully compatible with Alliance Series Design Implementation tools. Only the Xilinx implementation tools and a third party simulation tool are needed in addition to FPGA Express to fully create and simulate a design. This appendix includes the following sections.

- "Additional Documentation"
- "Alliance FPGA Express/Xilinx Design Flow"
- "Installing FPGA Express"
- "Entering a Design"
- "Simulating a Design"
- "Timing Constraints"
- "Porting Code from FPGA Compiler to FPGA Express"
- "Using LogiBLOX with FPGA Express"

Additional Documentation

The following documentation is available for FPGA Express and the Alliance Series Design Implementation tools for the current release of software.

- For installation of the Alliance Series Design Implementation Tools, refer to the *Alliance Series 2.1i Release Notes and Installation Guide*.
- For installation of FPGA Express, HDL-entry flow, and mixed entry flows, refer to the *FPGA Express User's Guide*, a hard copy document included with your FPGA Express software from Synopsys.
- For additional information on FPGA Express and the Xilinx flow, refer to the *Synopsys FPGA Express Design Guide,* available via ftp://ftp.xilinx.com/pub/swhelp/synopsys/xprsgde.zip. This file is a Word for Windows (95) version 7.0 file.

Alliance FPGA Express/Xilinx Design Flow

FPGA Express is the top-level design tool in the design flow. FPGA Express writes out an XNF file (EDIF for Virtex) which is fully compatible with the Alliance Series Design Implementation tools. The XNF file written out by FPGA Express can be accepted by NGDBuild or the Design Manager for creation of a PROM file.

The following types of simulation are possible with FPGA Express.

- Behavioral
- Post-NGDBuild
- Post-Map
- Post-synthesis post-route timing simulation (post-PAR)

For more specific information on simulation with FPGA Express, refer the *FPGA Express Design Guide*.

Refer to the following figure for a graphic representation of the design flow.

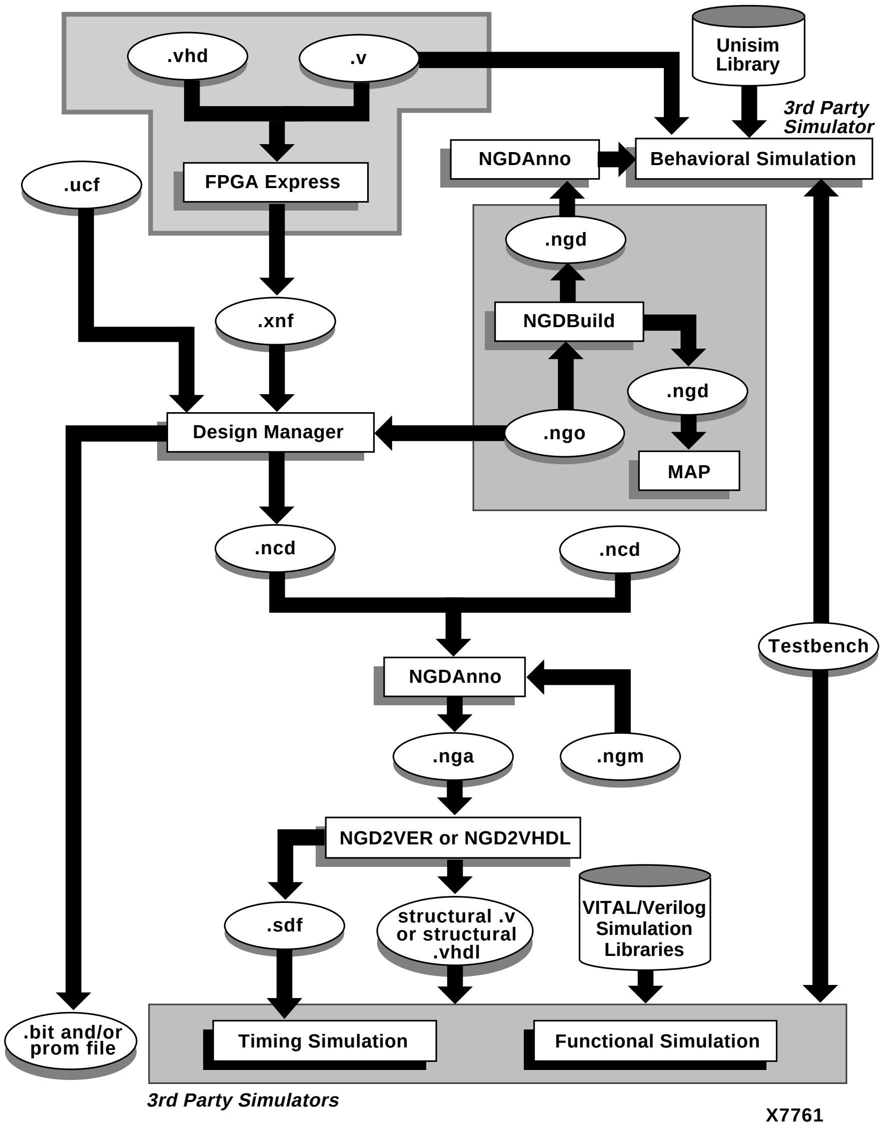


Figure A-1 Alliance FPGA Express/Xilinx Design Flow

Installing FPGA Express

Insert the FPGA Express CD into your CD-ROM drive. Start the Explorer and double-click on the CD-ROM icon. Double-click on setup.exe to start the install process.

For additional instructions on how to install FPGA Express on Windows 95 or Windows NT, refer to the *FPGA Express User's Guide* included with the FPGA Express software from Synopsys.

Entering a Design

To enter a design, use the following steps.

1. Start FPGA Express by selecting the following.

   `Program` → **`Synopsys`** → **`FPGA Express`**

2. Use a text editor to enter your design in Verilog or VHDL.
3. Define your project in FPGA Express by selecting.

   `File` → **`New...`**

4. Identify the HDL files for synthesis by selecting the following.

   `Synthesis` → **`Identify Sources`**

5. Specify the top-level file in your project by selecting the top-level file in the top-level design drop-down list in the middle of the FPGA Express toolbar.
6. Create an implementation by selecting the following.

   `Synthesis` → **`Create Implementation`**

7. Optimize your design by selecting the following.

   `Synthesis` → **`Optimize Chip`**

8. Write an XNF file by selecting the following.

   `Synthesis` → **`Export Netlist`**

Verilog or VHDL designs are the input files for the FPGA Express design flow, and the output is an XNF file (EDIF for Virtex designs), which can be processed directly by the Xilinx implementation tools. For details on defining projects in FPGA Express, entering HDL code, defining constraints in FPGA Express, supported devices, and design issues, refer to the *FPGA Express User's Guide* included with your FPGA Express software from Synopsys.

Simulating a Design

FPGA Express is a synthesis tool only. Simulation of designs with FPGA Express must be done with a third party simulation tool. For more information on simulation with FPGA Express, refer to the documentation of your third party simulation tool.

For VHDL simulation, the Xilinx VITAL libraries are required. The Xilinx VITAL libraries are located in the $XILINX/vhdl directory, ($XILINX is where the Xilinx software is installed). For Verilog simulation, the Xilinx Verilog libraries are required. The Xilinx Verilog libraries are located in the $XILINX/verilog directory.

For more information on the HDL simulation flow with FPGA Express, refer the *Development System Reference Guide*. For information on using the Design Manager in HDL simulation, refer to the *Design Manager/Flow Engine Guide*.

Note: There are three types of simulation possible behavioral, post-NGDBuild, and back-annotated timing simulation.

Timing Constraints

FPGA Express automatically inserts timespecs into the XNF file it writes out. For Virtex designs, a separate .ncf file containing timing constraints is created along with the .edf file. Optionally, the user can choose not to write out timespecs in the XNF file from FPGA Express. Instead, you can write the constraints in a .ucf file. The timespecs created by FPGA Express in the XNF file have the FROM: TO syntax.

Note: For more information on constraints and FPGA Express, refer to the *FPGA Express Expert Journal* at http://www.xilinx.com.

Porting Code from FPGA Compiler to FPGA Express

Read this section if you are porting a design from FPGA/Design Compiler to FPGA Express. If you are compiling a design originally compiled with FPGA/Design Compiler and the code is one hundred percent behavioral, then no modification of the code is needed. But, if you have instantiated components from the XSI libraries, some of these components do not exist in the FPGA Express libraries.

Some of the components that can be instantiated in the Xilinx design flow cannot be instantiated in the FPGA Express tool, since there are slight differences in names. For example, the BUFGP_F in the XSI component library does not exist in the FPGA Express component library. In FPGA Express, the equivalent name of the BUFGP_F is BUFGP. For a complete listing of the library cells that can be instantiated in FPGA Express, refer to the contents of the following.

```
fpgaexpress/lib/xc3000
fpgaexpress/lib/xc4000e
fpgaexpress/libxc5200
```

The fpgaexpress directory is where FPGA Express is installed on your system. In these directories, there are files with a .dsn extension. The string in front of .dsn is the name of the CELL that can be instantiated in FPGA Express.

In general, instantiation is not necessary. For the XC4000EX/XL/XLA/XV FPGA Express flow, you must instantiate the following components.

- I/O muxes
- Fast capture latches
- RAM
- BSCAN
- LogiBLOX

For examples of instantiation of I/O muxes, fast capture latches, RAM, and BSCAN, refer to the *“Xilinx Synopsys Interface Notes” appendix*.

Using LogiBLOX with FPGA Express

For information on using LogiBLOX and FPGA Express, refer to the FPGA Express “Tech Tips” at http://support.xilinx.com.

Appendix B

Mentor Graphics Interface Notes

This appendix describes how to set up the Mentor Graphics interface and associated libraries, and includes examples on pin locking and timing constraints. This chapter includes the following sections.

- "Additional Documentation"
- "Setting Up the Xilinx/Mentor Interface"
- "Mentor/Xilinx Software Design Flow"
- "Translating a Design to Xilinx EDIF"
- "Timing Simulation"
- "Mentor Interface Environment Variables"
- "Library Locations and Sample MGC Location Map"
- "Pin Locking"
- "Timing Constraints"

Additional Documentation

The following documentation is available for the Mentor Graphics interface.

- *Mentor Graphics Interface Guide* is available online. This manual describes installation setup and details how to use the Mentor Graphics Interface
- Mentor Graphics software documentation (for applications such as Design Architect, QuickSim, QuickHDL, and DVE) is available online and viewable with the Mentor-supplied BOLD Browser.

Setting Up the Xilinx/Mentor Interface

The following environment variables must be modified or added to run the Xilinx / Mentor interface tools.

- MGC_HOME (add)
- LCA (add)
- SIMPRIMS (add)
- MGC_GENLIB (add)
- MGC_LOCATION_MAP (add)
- path (modify)
- LD_LIBRARY_PATH (modify for Solaris)
- SHLIB_PATH (modify for HP-UX)

Set these variables as follows.

```
setenv MGC_HOME <installation_path_to_mentor>

setenv LCA $XILINX/mentor/data

setenv MGC_GENLIB $MGC_HOME/gen_lib

setenv MGC_LOCATION_MAP <location_of_actual_map_file>

set path = ($XILINX/mentor/bin/<platform_name> \

             $path)
```

For Solaris only.

```
setenv LD_LIBRARY_PATH $MGC_HOME/shared/
lib:$MGC_HOME/lib:$LD_LIBRARY_PATH
```

For HP-UX only.

```
setenv SHLIB_PATH $MGC_HOME/shared/lib:$MGC_HOME/
lib:$SHLIB_PATH
```

Following is an example of how to set your environment variables.

```
setenv MGC_HOME /usr/mentor

setenv LCA $XILINX/mentor/data

setenv SIMPRIMS $LCA/simprims

setenv MGC_GENLIB $MGC_HOME/gen_lib
setenv MGC_LOCATION_MAP /usr/data/mgc_location_map

set path = ($XILINX/mentor/bin/sol $path)

setenv LD_LIBRARY_PATH $MGC_HOME/shared/
lib:$MGC_HOME/lib:$LD_LIBRARY_PATH
```

Note: The previous settings assume that the Xilinx environment variables point to the appropriate area, as described in the Software Variable setup section of the *Alliance 2.1i Installation Guide and Release Notes*.

Mentor/Xilinx Software Design Flow

The following figure illustrates the Mentor Graphics and Xilinx software design flow. Shown are design entry, functional simulation, implementation, and timing simulation.

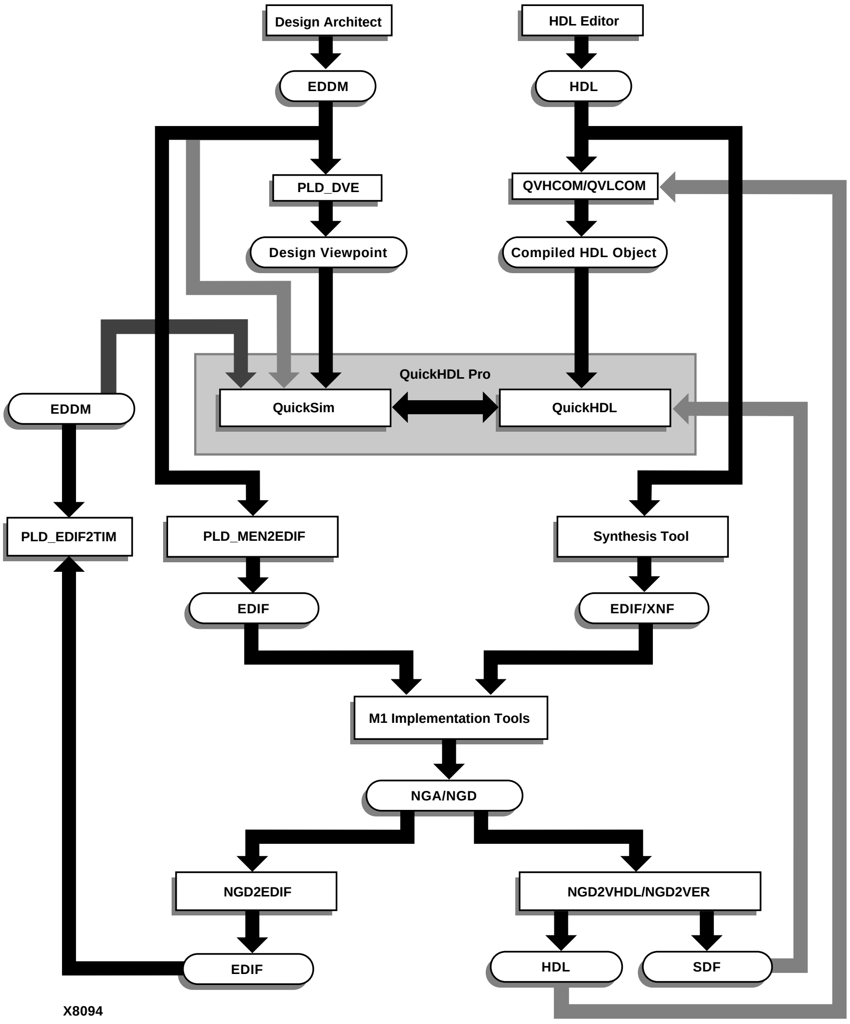


Figure B-1 Mentor/Xilinx Software Flow

- The design flow starts with design entry with PLD_DA (the Mentor schematic design tool).
- The design is processed by PLD_DVE to generate a Xilinx-style design viewpoint.

- The design is then passed to PLD_QuickSim for functional simulation.
- Once the design logic has been verified, the Mentor schematic is processed by PLD_MEN2EDIF to create an EDIF file.
- The EDIF file is passed to the Xilinx tools for implementation.
- The Xilinx tools create an EDN file that is processed by PLD_EDIF2TIM to create a timing-annotated EDDM netlist.
- This new netlist is processed by PLD_DVE to generate a Xilinx style design viewpoint.
- The design is passed to PLD_QuickSim to run in cross-probing mode for timing simulation.

For functional simulation, first generate a simulation viewpoint, with PLD_DVE. For example, to generate a viewpoint for the XC4000EX component my_design, use the following command.

```
pld_dve -s my_design xc4000ex
```

A specific viewpoint name can optionally be given after the technology type. If one is not given, a default viewpoint is created with the name *default*.

To simulate this design, use the following command.

```
pld_quicksim my_design
```

This runs QuickSim for functional simulation without cross-probing.

You may also use the PLD_DVE and PLD_QuickSim icons in PLD_DMGR. For more information on functional simulation, see the *Mentor Graphics Interface Guide.*

Translating a Design to Xilinx EDIF

To translate a design into an EDIF file for the Xilinx implementation tools, use the PLD_MEN2EDIF command. For example, to target my_design to the XC4000EX.

```
pld_men2edif my_design xc4000ex
```

You may also specify a viewpoint name after the technology type. If a viewpoint name is not given, a default viewpoint is used with the name *default*. This default viewpoint name is the same as that used by PLD_DVE.

You may also use the pld_men2edif icon in PLD_DMGR. For more information on PLD_MEN2EDIF, see the *Mentor Graphics Interface Guide.*

Timing Simulation

After implementing your design and generating an annotated NGA netlist (with NGDANNO), you must use NGD2EDIF to generate a timing-annotated EDIF netlist that Mentor can use.

Generating a Timing-Annotated EDIF Netlist

Use NGD2EDIF to generate a timing-annotated EDIF netlist. In the case of my_design, for example, enter the following.

```
ngd2edif -v mentor my_design.nga my_design.edn
```

This creates an EDN file compatible with the Mentor interface.

Generating a Timing Model

After creating the EDN file, run PLD_EDIF2TIM to generate a timing model with the following command.

```
pld_edif2tim my_design.edn
```

This creates an EDDM-type component under my_design_lib/ my_design, as well as a simulation viewpoint for that component.

Running PLD_QuickSim

After generating the simulation viewpoint, run PLD_QuickSim with cross-probing on this new component. (If you do not wish to annotate simulation values onto your original schematic, you may remove the -cp option to run without cross-probing.)

```
pld_quicksim my_design_lib/my design

-cp -tim type -consm messages
```

QuickSim will start up and read in the new timing-annotated EDDM netlist. DVE will also start up. Open the viewpoint and schematic sheet for your *original* schematic in DVE to annotate simulation values (from QuickSim) onto that front-end schematic.

You may also use the PLD_EDIF2TIM and PLD_QuickSim icons in PLD_DMGR. For more information on timing simulation, including a more detailed explanation on cross-probing, see the *Mentor Graphics Interface Guide.*

Mentor Interface Environment Variables

Set the following environment variables.

```
setenv LCA $XILINX/mentor/data
setenv SIMPRIMS $LCA/simprims
set path = ($XILINX/mentor/bin/sol $path)
```

(This example is for Solaris workstations. Replace "sol" with "hp" for HP-UX workstations.) These variables are in addition to the XILINX environment variable settings required by the Alliance Series Design Implementation Tools. To refer to the Mentor-specific variables such as MGC_HOME and MGC_LOCATION_MAP, see the *Mentor Graphics Interface Guide* for more information.

Library Locations and Sample MGC Location Map

All Xilinx libraries reside under the $LCA directory as with XACT 5.x. Also underneath this directory is the "simprims" (simulation primitives) library that QuickSim must use to simulate back-end timing simulation models. This requires your MGC location map to have the following lines in addition to any other soft names (including MGC_GENLIB) you have included.

```
MGC_LOCATION_MAP_1

$LCA
(blank line)

$SIMPRIMS
(blank line)
```

As always, your $MGC_LOCATION_MAP file points to the location of this file. For more information on location maps, see the *Mentor Graphics Interface Guide*.

Pin Locking

Pad symbols (IPAD, OPAD, etc.) have generic pin-location ("LOC") properties already attached to them. (They appear as "PXX" on the pad symbol.) You can place pads in specific locations on the device by modifying these properties as required. (An example property value for a pad symbol may be “P24”.) Note that “bused” pad symbols (for example, IPAD8) may take a comma-separated list (in MSB to LSB order) of locations (P24, P23, P22, . . .). For more information on location constraints, see the *Libraries Guide*.

Timing Constraints

Timing constraints can be placed as properties on a TIMESPEC symbol in the design. The Timespec label (the label that begins with “TS”) is entered as the property name, while the timing specification (for example, “FROM:FFS:TO:FFS=30NS”) is entered as the property value. For more information on timing constraints, see the *Development System Reference Guide*.

Appendix C

Xilinx Synopsys Interface Notes

This appendix provides information on setting up the Xilinx Synopsys Interface (XSI) and associated libraries. Example files are included to help you set up the FPGA Compiler and VSS with the Xilinx software. This chapter contains the following sections.

- “Documentation”
- “Setting Up the Synopsys Interface”
- “Examples of Synopsys Setup Files”
- “Entity Coding Examples”

Documentation

The following documentation is available for the Synopsys interface.

- The *Xilinx Synopsys Interface Guide* is available on the Alliance 2.1i Documentation CD-ROM.
- Alliance 2.1i *Release Documentation* describes installation setup and current issues regarding the use of the Synopsys interface.
- For converting an XACT 5.x.x Synopsys design to M1, refer to the *Xilinx Software Conversion Guide from XACTstep v5.X.X to XACT-step vM1.X.X*.

Setting Up the Synopsys Interface

The following environment variables must be modified or added to run the Synopsys interface tools.

- SYNOPSYS (add)
- PATH (modify)

- LD_LIBRARY_PATH (modify)
- SHLIB_PATH (modify)

Set these variables as follows.

setenv SYNOPSYS *installation_path_to_synopsys*

set path = ($XILINX/bin/*platform_name* \

$SYNOPSYS/*platform_name***/syn/bin \**

$SYNOPSYS/*platform_name***/sim/bin \**

$path)

For Solaris only.

setenv LD_LIBRARY_PATH $SYNOPSYS**/*platform_name***/sim/lib:$LD_LIBRARY_PATH

For HP/UX only.

setenv SHLIB_PATH $SYNOPSYS**/*platform_name***/sim/lib:$SHLIB_PATH

The following is an example.

```
setenv SYNOPSYS /usr/synopsys
set path = ($XILINX/bin/sol \
$SYNOPSYS/sol/syn/bin \
$SYNOPSYS/sol/sim/bin \
$path)
setenv LD_LIBRARY_PATH $SYNOPSYS/sol/sim/
lib:$LD_LIBRARY_PATH
```

Setting up the XDW and Simulation Libraries

Note: If you are not using FPGA CompilerII v3.2 or a later, you must re-compile the Xilinx DesignWare (XDW) libraries.

The XSI (Xilinx Synopsys Interface) simulation and XDW (Xilinx DesignWare) libraries are compiled for Synopsys v1998.08. If you are using the latest version of XSI with a version of Synopsys newer than v1998.08, you must re-compile the XDW libraries with the version of

Synopsys you are using. If you are going to simulate with VSS, you must re-compile the simulation libraries.

Compiling the libraries in the $XILINX area requires write permissions to this area. If you copy the $XILINX/synopsys directory to a local area, you do not need rewrite permissions for the $XILINX area to re-compile the libraries. However, verify that the .synopsys_dc.setup and .synopsys_vss.setup files use the local copy.

Compiling XDW Libraries

Follow these steps to compile the XDW libraries.

1. Set up your Xilinx and Synopsys software environments.
2. Go to the $XILINX/synopsys/libraries/dw/src directory.
3. In this directory, there are ten subdirectories that represent the Xilinx device families that have XDW libraries. If you are going to use any of the device families listed, you must go to the corresponding subdirectory and run the .dc compile script in that directory. For example, for a Spartan device, enter the following commands.

   ```
   cd spartan
   ```

   Run the install_dw.dc script by entering the following.

   ```
   dc_shell -f install_dw.dc
   ```
4. When the script is finished, go back to $XILINX/synopsys/libraries/dw/src. Repeat these steps for each device you plan on using.

Compiling the Simulation Libraries

Note: The following procedure is for compiling the XSI simulation libraries with VSS. If you are using a different HDL simulator, refer to your simulator’s documentation for instructions on compiling HDL simulation libraries.

1. Setup your XSI and Synopsys software environments.
2. Go to the $XILINX/synopsys/libraries/sim/src directory.
3. In this directory, there are subdirectories that represent the five simulation libraries, described as follows.

- LogiBLOX — for functional simulation of VHDL designs with instantiated LogiBLOX components
- SimPrims — timing simulation library
- UNISIMS — functional simulation library
- XC9000 — XC9500 functional simulation library
- XDW — Functional simulation library for post-synthesis (FPGA compiler) pre-NGDBuild simulation

Some or all of these libraries need to be re-compiled depending on the device and type of simulation you plan on using. Xilinx recommends compiling the logiblox, simprims, and unisims libraries. Use the following steps to compile these libraries.

4. Go to the logiblox directory and enter the following command.

   ```
   ./analyze.csh
   ```

   Go back to the $XILINX/synopsys/libraries/sim/src directory.

5. Go to the simprims directory and enter the following command.

   ```
   ./analyze.csh
   ```

   Go back to the $XILINX/synopsys/libraries/sim/src directory.

6. Go to the unisims directory and enter the following command.

   ```
   ./analyze.csh
   ```

   The unisims directory also contains the analyze_52k.csh script. If you plan on simulating XC5200 devices, you must run this script as well. You must also edit the .synopsys_dc.setup file in the unisims directory to point to a location for the compiled XC5200 libraries.

   Go back to the $XILINX/synopsys/libraries/sim/src directory.

7. Go to the xdw directory and enter the following command.

   ```
   ./analyze.csh
   ```

   Go back to the $XILINX/synopsys/libraries/sim/src directory.

8. Go to the xc9000/ftgs directory and enter the following command.

   ```
   dc_shell -f install_xc9000.dc
   ```

Examples of Synopsys Setup Files

This section includes examples of the Synopsys setup files needed to run the FPGA Compiler and VSS with the Xilinx tools. These examples are for XC4000XL and Virtex devices. Other FPGA and CPLD templates are in the Xilinx installation path, $XILINX/synopsys/examples.

XC4000 Devices

Although the following .synopsys_dc.setup file example is for an XC4000XL device, it is similar to the setup file required for XC4000E/EX/XLA/XV devices.

Example .synopsys_dc.setup File

```
/* Template .synopsys_dc.setup file for Xilinx */
/* For targeting a XC4000XL                   */
XilinxInstall = get_unix_variable(XILINX);
SynopsysInstall = get_unix_variable(SYNOPSYS);
search_path = { . \
XilinxInstall + /synopsys/libraries/syn \
SynopsysInstall + /libraries/syn }
/* Define a work library.You must create ‘work’  */
define_design_lib WORK -path ./WORK
/* Declare the Xilinx DesignWare library         */
define_design_lib xdw_4000xl -path \
XilinxInstall + /synopsys/libraries/dw/lib/xc4000xl

/* General configuration settings.              */
compile_fix_multiple_port_nets = true
xnfout_constraints_per_endpoint = 0
xnfout_library_version = "2.0.0"

bus_naming_style = "%s<%d>"
bus_dimension_separator_style = "><"
bus_inference_style = "%s<%d>"
/*     synlibs -fc 4028ex-3 >> .synopsys_dc.setup */
```

Example .synopsys_vss.setup File

```
/* Set any simulation preferences.              */
TIMEBASE        = NS
TIME_RES_FACTOR = 0.1
/* Define a work library in the current project */
```

```
WORK    > DEFAULT
DEFAULT : ./WORK
/* Set up SIMPRIM Back-annotation libraries     */
SIMPRIM : $XILINX/synopsys/libraries/sim/lib/simprims
/* Set up LogiBLOX simulation libraries         */
LOGIBLOX : $XILINX/synopsys/libraries/sim/lib/logiblox
/* Set up example pointers to the Xilinx Unisim functional simulation
library */
UNISIM: $XILINX/synopsys/libraries/sim/lib/unisims
```

Example Script File for XC4000E/EX/XL/XV Designs

This section describes the typical sequence of commands used to process designs with the Synopsys interface. You should run the commands at the dc_shell command line, either individually or in batch mode. While every design may not require all the commands used in this section, the following example represents a good starting point for most designs. This script file includes information on I/O pin location constraints, timing constraints, setting the part-type, controlling I/O characteristics, and controlling Synopsys mapping and packing functions.

```
/* Sample Script for Synopsys to Xilinx Using */
/* FPGA Compiler targeting a XC4000EX device*/
/* Set the name of the design's top-level */
TOP = <design_name>
designer = "XSI Team"
   company  = "Xilinx, Inc"
   part     = "4028expg299-3"
/* Analyze and Elaborate the design file. */
analyze -format vhdl TOP + ".vhd"
elaborate TOP
/* Set the current design to the top level. */
current_design TOP
/* Set the synthesis design constraints. */
remove_constraint -all
   /* Some example constraints */
   create_clock <clock_port_name> -period 50
   set_input_delay 5 -clock <clock_port_name> \
     { <a_list_of_input_ports> }

   set_output_delay 5 -clock <clock_port_name> \
     { <a_list_of_output_ports> }

   set_max_delay 100 -from <source> -to <destination>
   set_false_path -from <source> -to <destination>
/* Indicate which ports are pads. */
set_port_is_pad “*”
   /* Some example I/O parameters */
   set_pad_type -pullup <port_name>
   set_pad_type -no_clock all_inputs()
   set_pad_type -clock <clock_port_name>
   set_pad_type -exact BUFGS_F <hi_fanout_port_name>
   set_pad_type -slewrate HIGH all_outputs()
insert_pads
/* Synthesize the design.*/
compile -boundary_optimization -map_effort med
/* Write the design report files. */
   report_fpga > TOP + ".fpga"
   report_timing > TOP + ".timing"
/* Write out an intermediate DB file to save state*/
write -format db -hierarchy -output TOP + "_compiled .db"
/* Replace CLBs and IOBs primitives (XC4000E/EX/XL only)*/
replace_fpga
/* reapply set_max_delay/set_false_path if using FPGA compiler */
/* Set the part type for the output netlist.
set_attribute TOP "part" -type string part
/* Optional attribute to remove the mapping symbols*/set_attribute
find(design,"*")\
"xnfout_write_map_ symbols" -type boolean FALSE
/* Add any I/O constraints to the design. */
set_attribute <port_name> "pad_location" \
-type string "<pad_location>"
/* Write out the intermediate DB file to save state*/
write -format db -hierarchy -output TOP + ".db"
/* Write out the timing constraints*/
ungroup -all -flatter
write_script > TOP + ".dc"
/* Save design in XNF format as <design>.sxnf */
write -format xnf -hierarchy -output TOP + ".sxnf"
/* Convert constraints to Xilinx syntax */
sh dc2ncf TOP + ".dc"
/* Exit the Compiler. */
exit

/* Now run the Xilinx design implementation tools. */
```

Virtex Devices

The following setup file examples are for Virtex devices.

Example .synopsys_dc.setup File

```
/* ==================================================  */
/* Template .synopsys_dc.setup file for Xilinx designs  */
/*      For use with Synopsys FPGA Compiler.          */
/* ==================================================  */

/* ================================================= */
/* The Synopsys search path should be set to point   */
/* to the directories that contain the various       */
/* synthesis libraries used by FPGA Compiler during  */
/* synthesis.                                       */
/* ================================================= */

XilinxInstall = get_unix_variable(XILINX);
SynopsysInstall = get_unix_variable(SYNOPSYS);

search_path = { .       \
        XilinxInstall + /synopsys/libraries/syn \
        SynopsysInstall + /libraries/syn }

                /* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! */
                /* Ensure that your UNIX environment */
                /* includes the two environment var- */
                /* iables: $XILINX (points to the    */
                /* Xilinx installation directory) and*/
                /* $SYNOPSYS (points to the Synopsys */
                /* installation directory.)          */
                /* !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! */

/* ================================================= */
/* Define a work library in the current project dir  */
/* to hold temporary files and keep the project area */
/* uncluttered. Note: You must create a subdirectory */
/* in your project directory called WORK.            */
/* ================================================= */


   define_design_lib WORK -path ./WORK


bus_extraction_style = "%s<%d:%d>"
bus_naming_style = "%s<%d>"
bus_dimension_separator_style = "><"

edifin_lib_logic_1_symbol = "VCC"
```

```
edifin_lib_logic_0_symbol = "GND"
edifout_ground_name = "GND"
edifout_ground_pin_name = "G"
edifout_power_name = "VCC"
edifout_power_pin_name = "P"
edifout_netlist_only = "true"
edifout_no_array = "false"
edifout_power_and_ground_representation = "cell"
edifout_write_properties_list = {"CLK1X_DUTY" "INIT_00" "INIT_01"
"INIT_02" "INIT_03" \
 "INIT_04"  "INIT_05" "INIT_06" "INIT_07" "INIT_08" "INIT_09" "INIT_0A"
"INIT_0B" "INIT_0C" \
 "INIT_0D" "INIT_0E" "INIT_0F" "INIT" "CLKDV_DIVIDE" "IOB" "EQN"
"lut_function"}

/* ================================================= */
/* Set the link, target and synthetic library        */
/* variables. Use synlibs to                        */
/* determine the link and target library settings.   */
/* You may like to copy this file to your project    */
/* directory, rename it ".synopsys_dc.setup" and     */
/* append the output of synlibs. For example:        */
/* synlibs xfpga_virtex-3 >> .synopsys_dc.setup      */
/* ================================================= */

link_library = {xfpga_virtex-5.db }
link_library = {xfpga_virtex-5.db }
symbol_library = {virtex.sdb}
define_design_lib xdw_virtex -path XilinxInstall + /synopsys/libraries/
dw/lib/virtex
synthetic_library = {xdw_virtex.sldb standard.sldb}
```

Example Script File for Virtex Devices

```
/* ==================================================  */
/*    Sample Script for Synopsys to Xilinx Using      */
/*                  FPGA Compiler                    */
/*                                                   */
/*  Targets the Xilinx XCV150PQ240-3 and assumes a    */
/*   VHDL source file by way of an example.           */
/*                                                   */
/*   For general use with VIRTEX architectures.       */
/* ================================================== */

/* ================================================= */
/* Set the name of the design’s top-level module.    */
/* (Makes the script more readable and portable.)    */
/* Also set some useful variables to record the      */
/* designer and company name.                       */
/* ================================================= */

   TOP = <design_name>
                       /* ========================== */
                       /* Note: Assumes design file- */
                       /* name and entity name are   */
                       /* the same (minus extension) */
                       /* ========================== */

   designer = "XSI Team"
   company  = "Xilinx, Inc"
   part     = "XCV150PQ240-3"

/* ================================================= */
/* Analyze and Elaborate the design file and specify */
/* the design file format.                          */
/* ================================================= */

   analyze -format vhdl TOP + ".vhd"

                     /* ============================ */
                     /* You must analyze lower-level */
                     /* hierarchy modules here       */
                     /* ============================ */
   elaborate TOP

/* ================================================= */
/* Set the current design to the top level.          */
/* ================================================= */

   current_design TOP

/* ================================================= */
/* Set the synthesis design constraints.             */
/* ================================================= */

   remove_constraint -all

/* If setting timing constraints, do it here.
   For example:                                      */
/*
   create_clock <clock_pad_name> -period 50
```

```
*/


/* ================================================= */
/* Indicate those ports on the top-level module that */
/* should become chip-level I/O pads. Assign any I/O */
/* attributes or parameters and perform the I/O      */
/* synthesis.                                       */
/* ================================================= */

   set_port_is_pad "*"
   set_pad_type -slewrate HIGH all_outputs()
   insert_pads

/* +++++++++++++++++++++++++++++++++++++++++++++++++ */
/*              Compile the design                  */
/* +++++++++++++++++++++++++++++++++++++++++++++++++ */

   compile -map_effort med

/* ================================================= */
/* Write the design report files.                   */
/* ================================================= */

   report_fpga > TOP + ".fpga"
   report_timing > TOP + ".timing"

/* ================================================= */
/* Set the part type for the output netlist.        */
/* ================================================= */

   set_attribute TOP "part" -type string part

/* ================================================= */
/* Save design in EDIF format as <design>.sedif     */
/* ================================================= */

   write -format xnf -hierarchy -output TOP + ".sedif"

/* ================================================= */
/* Write out the design to a DB.                    */
/* ================================================= */

   write -format db -hierarchy -output TOP + ".db"
```

```
/* ================================================= */
/* Write-out the timing constraints that were        */
/* applied earlier. (Note that any design hierarchy  */
/* needs to be flattened before the constraints are  */
/* written-out.)                                    */
/* ================================================= */

   write_script > TOP + ".dc"

/* ================================================= */
/* Call the Synopsys-to-Xilinx constraints translator*/
/* utility DC2NCF to convert the Synopsys constraints*/
/* to a Xilinx NCF file. You may like to view        */
/* dc2ncf.log to review the translation process.     */
/* ================================================= */

   sh dc2ncf -w TOP + ".dc"

/* ================================================= */
/* Exit the Compiler.                               */
/* ================================================= */

   exit

/* ================================================= */
/* Now run the Xilinx design implementation tools.   */
/* ================================================= */
```

Entity Coding Examples

This section includes VHDL and Verilog code examples.

VHDL

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

use IEEE.STD_LOGIC_ARITH.all;

use IEEE.STD_LOGIC_UNSIGNED.all;

entity example is

port(RAMOUT:out STD_LOGIC; DIN: in STD_LOGIC;

AD4,AD3,AD2,AD1,AD0,RMWE,RMWCLK: in STD_LOGIC;
```

```
REG1OUT: out STD_LOGIC; DTA1,CLK1: in STD_LOGIC;
REG2OUT: out STD_LOGIC; DTA2,CLK2: in STD_LOGIC;
LTCHOUT: out STD_LOGIC;
LTD,LTGF,LTGE,LTCLK: in STD_LOGIC;
FASTOUT: out STD_LOGIC; FASTIN: in STD_LOGIC;
MUXOUT: out STD_LOGIC; MUXIN1,MUXIN2: in STD_LOGIC);
end example;
architecture inside of example is
signal ground: STD_LOGIC;
component RAM32X1S
port(O: out STD_LOGIC; D: in STD_LOGIC;
A4,A3,A2,A1,A0,WE,WCLK: in STD_LOGIC);
signal rstsig: STD_LOGIC;
end component;
component IFD_F
port(Q: out STD_LOGIC; D,C: in STD_LOGIC);
end component;
component OFD_F
port(Q: out STD_LOGIC; D,C: in STD_LOGIC);
end component;
component STARTBUF
port(GSRIN: in STD_LOGIC;
    (GSTIN: in STD_LOGIC;
    (CLKIN: in STD_LOGIC;
    (GSROUT: out STD_LOGIC;
D,GF,CE,C: in STD_LOGIC);
end component;
component BUFFCLK
port(O: out STD_LOGIC; I: in STD_LOGIC);
end component;
```

```
component OAND2

port(O: out STD_LOGIC; F,I0: in STD_LOGIC);

end component;

begin

U0: RAM32X1S port map(O=>RAMOUT,D=>DIN,

A4=>AD4,A3=>AD3,A2=>AD2,A1=>AD1,A0=>AD0,WE=>RMWE,WCLK=>RMWCLK);

U1: IFD_F port map(Q=>REG1OUT,D=>DTA1,C=>CLK1);

U2: OFD_F port map(Q=>REG2OUT,D=>DTA2,C=>CLK2);

U3: STARTBUF port map
(GSRIN=>RESET,GTSIN=>GROUND,CLKIN=>GROUND,GSROUT=>RSTSIG);

U4: BUFFCLK port map(O=>FASTOUT,I=>FASTIN);

U5: OAND2 port map(O=>MUXOUT,F=>MUXIN1,I0=>MUXIN2);

end inside;
```

Verilog Code: Module Example

```
module example ( RAMOUT,DIN,AD,RMWE,RMWCLK,
REG1OUT,DTA1,CLK1,REG2OUT,DTA2,CLK2,
LTCHOUT,LTD,LTGF,LTGE,LTCLK,
FASTOUT,FASTIN,MUXOUT,MUXIN1,MUXIN2);

input RMWE,RMWCLK,DIN,DTA1,CLK1,DTA2,CLK2,LTD,LTGF,LTGE,LTCLK

input FASTIN,MUXIN1,MUXIN2;

input [4:0] AD;

output RAMOUT,REG1OUT,REG2OUT,LTCHOUT,FASTOUT,MUXOUT;

RAM32X1S U0
(.O(RAMOUT),.D(DIN),.A4(AD[4]),.A3(AD[3]),.A2(AD[2]),.A1(AD[1]),.A0(AD[0]
),.WE(RMWE),.WCLK(RMWCLK));

IFD_F U1 (.Q(REG1OUT),.D(DTA1),.C(CLK1));

OFD_F U2 (.Q(REG2OUT),.D(DTA2),.C(CLK2));

BUFFCLK U4 (.O(FASTOUT),.I(FASTIN));

OAND2 U5 (.O(MUXOUT),.F(MUXIN1),.I0(MUXIN2));

endmodule
```

Comments About Code

When instantiating IOB components such as IFD_F, OFD_F, ILFFX, BUFFCLK, or OAND2, make sure that unnecessary IBUF/OBUF/OBUFTs are not inserted. Remove the port_is_pad attribute from the pin that is directly connected to a pad, such as the D pin of the IFD_F, or the .D pin of the ILFFX. To remove the port_is_pad attributes, use the remove_attribute command.

Appendix D

Viewlogic Interface Notes

This appendix provides information on setting up the Viewlogic interface and project libraries. Included are examples for assigning location constraints and timing specifications. The following sections are included in this chapter.

- "Documentation"
- "Setting Up Viewlogic Interface on Workstations"
- "Setting Up Viewlogic Interface on the PC"
- "Setting Up Project Libraries"
- "Assigning a Pin Location"

Documentation

The following documentation is available for the Viewlogic interface.

- The *Viewlogic Interface Guide* is available on-line.
- The *Alliance 2.1i Release Notes and Installation Guide* describes installation setup and current issues regarding the use of the Viewlogic interface.

Setting Up Viewlogic Interface on Workstations

The following environment variables must be modified or added to run the Viewlogic tools.

- POWERVIEW (add)
- WDIR (add)
- VANTAGE_VSS (add)
- PATH (modify)

- LM_LICENSE_FILE (modify)
- LD_LIBRARY_PATH (modify, Solaris only)
- SHLIB_PATH (modify, HP-UX only)

In addition to the variables set for the Xilinx implementation tools, these variables should be set as follows.

```
setenv POWERVIEW <installation_path_to_viewlogic>
setenv WDIR $XILINX/viewlog/data/logiblox/standard:$POWERVIEW/standard
setenv VANTAGE_VSS $POWERVIEW/standard/van_vss
set path = ($POWERVIEW \
             $VANTAGE_VSS/pgm/dir   \
             $path)
setenv LM_LICENSE_FILE <path_to_viewlogic_license_file>:$LM_LICENSE_FILE
```

For Solaris only.

```
setenv LD_LIBRARY_PATH $POWERVIEW/standard/fusion:$LD_LIBRARY_PATH
```

For HP/UX only.

```
setenv SHLIB_PATH $POWERVIEW/standard/fusion:$SHLIB_PATH
```

Note: The previous settings assume that the XILINX, LD_LIBRARY_PATH, SHLIB_PATH, and LM_LICENSE_FILE environment variables have been previously assigned to point to the appropriate areas. The POWERVIEW variable is not required by Xilinx or by the Viewlogic software. It is used to simplify these environment variable settings.

Setting Up Viewlogic Interface on the PC

The following environment variables must be modified or added to run the Viewlogic interface tools on a PC.

- PATH(modify)
- WDIR (new)
- VANTAGE_VSS (new)
- VANTAGE_CC (new)
- LM_LICENSE_FILE (modify)

These variables are modified/added in the following manner by the Workview Office installation software. The following examples assume that all the software has been installed to the default locations on the C:\ drive. If these default paths have been changed, the environment settings must change accordingly.

```
PATH=C:\WVOFFICE;%PATH%

SET WDIR=C:\WVOFFICE\STANDARD

SET VANTAGE_VSS=C:\WVOFFICE\V

SET VANTAGE_CC=C:\WVOFFICE\CL

SET
LM_LICENSE_FILE=C:\WVOFFICE\STANDARD\LICENSE.DAT,;
C:XILINX\DATA\LICENCE.DAT
```

Note: The LM_LICENSE_FILE must be set exactly as shown, with a comma and semicolon(,;) between the two paths if Workview Office 7.31 or older is used. This is because Viewlogic and Xilinx use different versions of Flex/LM licensing which use different delimiters in this variable. The comma is not required for Workview Office 7.4 or newer.

For Windows NT 4.0 users only, select the following.

`Start` → **`Settings`** → **`Control Panel`**

Double click on the System icon and select the Environment tab. Verify the settings previously shown are listed in either the System Variables section or the User Variables section. They do not appear exactly as shown in the previous example; the variable is shown under the Variable header and the path is shown under the Value header. The word “set” does not appear.

For Windows 95 users only, run SYSEDIT to open the AUTOEXEC.BAT file, and verify the environment settings are as previously shown.

Setting Up Project Libraries

This section describes project library setup for the workstation and the PC.

Workstation

The first step before creating or modifying a design in ViewDraw is to set up the project libraries. When creating a Viewlogic design to be processed by the Xilinx Alliance Series Design Implementation Tools, the Unified Libraries must be used. These libraries must be defined in the viewdraw.ini file located in the project's working directory.

To define a library in the viewdraw.ini, it must be added to the search order at the end of the viewdraw.ini file.

The Xilinx Libraries for use with Viewlogic schematic entry tools are located in $XILINX/viewlog/data. Directories exist for all the supported Xilinx families as well as LogiBLOX and the required Simprims, Builtin, and Xbuiltin libraries.

The following example is a library search order needed to create an XC4000EX design.

```
dir [p] . (primary)
dir [rm] /tools/xilinx/viewlog/data/xc4000x (xc4000x)
dir [r] /tools/xilinx/viewlog/data/logiblox (logiblox)
dir [rm] /tools/xilinx/viewlog/data/simprims (simprims)
dir [rm] /tools/xilinx/viewlog/data/builtin (builtin)
dir [rm] /tools/xilinx/viewlog/data/xbuiltin (xbuiltin)
```

Note: The XC4000X library and alias were new with the 1.4 version of the Xilinx software. This library is used for all XC4000EX/XL/XV designs. To use the 1.4 libraries with designs created with previous versions of the software, add the following line to the viewdraw.ini file before the LogiBLOX line.

```
dir [rm] /tools/xilinx/viewlog/data/xc4000x (xc4000ex)
```

Following are the features of this search order.

- The LogiBLOX library replaces X-BLOX. This library is read-only and not in megafile format.
- There is a new library, “Simprims,” that is used only for simulation.
- Order counts; user and family libraries must appear before Simprims, Builtin, and Xbuiltin.

- Full paths must be used; do not use $XILINX to abbreviate the path.

Xilinx Commands in ViewDraw

Once the environment variables have been set and the libraries have been defined, you may begin your schematic design work. The one Xilinx feature within ViewDraw is the addition of two new commands under the pull-down menus.

- Add → LogiBLOX is used to create a new LogiBLOX component.
- Change → LogiBLOX is used to modify an existing LogiBLOX component.

PC

The first step before creating or modifying a design in ViewDraw is to set up the project libraries. When creating a Viewlogic design to be processed by the M1 Alliance Series Design Implementation Tools, the M1 Libraries must be used. These libraries must be defined in the Viewlogic project file (VPJ).

Note: Use the Workview Office Project Manager to make any modifications to the project libraries. Do not directly modify the viewdraw.ini file.

The Xilinx Libraries for use with Viewlogic schematic entry tools are located in C:\XILINX\VIEWLOG\DATA. Directories exist for all the supported Xilinx families as well as LogiBLOX and the required Simprims, Builtin, and Xbuiltin libraries.

The following is the library search order needed to create an XC4000EX design.

```
dir [p] . (primary)

dir [rm] C:\xilinx\viewlog\data\xc4000x (xc4000x)

dir [r] C:\xilinx\viewlog\data\logiblox (logiblox)

dir [rm] C:\xilinx\viewlog\data\simprims (simprims)

dir [rm] C:\xilinx\viewlog\data\builtin (builtin)

dir [rm] C:\xilinx\viewlog\data\xbuiltin (xbuiltin)
```

For information about how to use the Workview Office Project Manager to define the project libraries, refer to the *Viewlogic Interface/ Tutorial Guide.*

Note: The XC4000X library and alias were new with the 1.4 version of the Xilinx software. This library is used for all XC4000EX / XL / XV designs. To use the 1.4 libraries with designs created with previous versions of the software, add the following line to the viewdraw.ini file before the LogiBLOX line.

```
dir [rm] C:\xilinx\viewlog\data\xc4000x (xc4000ex)
```

Features of this search order.

- The LogiBLOX library replaces X-BLOX. This library is read-only and not in megafile format.
- There is a new library, “Simprims”, that is used only for simulation.
- Order counts; user and family libraries must appear before Simprims, Builtin, and Xbuiltin.
- Full paths must be used; do not use %XILINX% to abbreviate the path.

Assigning a Pin Location

To assign the location of a pin to a specific pad location, simply add a location constraint attribute to that pad on the schematic.

1. Select the IPAD, OPAD or IOPAD you wish to constrain.
2. For workstation users, select **Change** → **`Attr`** → **`Dialog`** → **`All`**. The Change Attributes dialog box will display.

   For PC users, double click on the pad.
3. Enter LOC in the Name field, and enter the pin instance in the Component Value field.

   Valid pin syntax for quad flat packages is P#, where # is the actual device pin number desired. For example: LOC = P11.

   Valid pin syntax for grid array packages is RC, where R is the actual row and C is the column of the device pin. For example: LOC = A13.

4. Click on **OK**. The LOC attribute will now be placed next to the pad.

Bus-wide pads (that is IPAD16) must be constrained within a user constraints file (.ucf).

Timing Constraints

Timing constraints may be placed via the TIMESPEC symbol in the design. The TIMESPEC symbol is found in the Xilinx family library (for example., XC4000X). After placing this symbol on the top level of your design, the timespecs are added as properties of this symbol. The Timespec label (the label that begins with "TS") is entered in the Name field, while the timing specification (e.g., "FROM:FFS:TO:FFS=30ns") is entered in the Value field.

For more information on this subject, refer to the *Viewlogic Interface Guide*. For more information on timing constraints, see the *Development System Reference Guide*.

Appendix E

Using LogiBLOX with CAE Interfaces

LogiBLOX is graphical design tool for creating high-level modules such as counters, shift registers and multiplexers. LogiBLOX includes both a library of generic modules and a set of tools for customizing these modules.

With LogiBLOX, high-level LogiBLOX modules that will fit into your schematic-based design, or HDL synthesis-based design can be created and processed. These modules can be used in designs generated with schematic editors from Mentor Graphics, Viewlogic and Xilinx Foundation Package, as well as third-party synthesis tools such as Synopsys FPGA Compiler/FPGA Express, and Exemplar.

Note: LogiBLOX supports XC3000A, XC3100A, XC4000E, XC4000L, XC4000EX, XC4000XL, XC4000XV, XC4000XLA, XC5200, XC9500, XC9500XL, Spartan, and Spartan XL.

For more information about the Core Generator system, refer to the *Core Generator User Guide*, or the Xilinx Web site. This GUI tool supports different devices than LogiBLOX.

This chapter includes the following sections.

- “Documentation”
- “Setting Up LogiBLOX on a Workstation”
- “Setting Up LogiBLOX on a PC”
- “Starting LogiBLOX”
- “Using LogiBLOX for Schematic Design”
- “Using LogiBLOX for HDL Synthesis Design”
- “Analyzing a LogiBLOX Module”
- “LogiBLOX Modules”

Documentation

The following documentation is available for the LogiBLOX program.

- The *LogiBLOX User Guide* is available on the CD-ROM supplied with your software.
- The LogiBLOX online help can be accessed from LogiBLOX, GUI.
- The Alliance 2.1i *Release Documentation* describes installation setup and current issues regarding the use of LogiBLOX.
- The *Xilinx Software Conversion Guide from XACTstep v5.X.X to XACTstep vM1.X.X* compares XBLOX and LogiBLOX, and how to convert an X-BLOX design to LogiBLOX. The *Xilinx Software Conversion Guide from XACTstep v5.X.X to XACTstep vM1.X.X* and other application notes can be found in the userware directory of the Xilinx CD, or at the Xilinx web site http://www.xilinx.com.

Setting Up LogiBLOX on a Workstation

This section describes the issuing commands and the file modifications required to set up your environment when using a third-party schematic design tool on a workstation. You must set up the Xilinx environment and interface environment as described in the Software Installation chapter of the *Alliance 2.1i Release Notes and Installation Guide*, and in the appropriate appendix in this manual.

Mentor Interface Environment Variables

To use LogiBLOX with Mentor, set the following environment variable.

```
setenv LCA $XILINX/mentor/data

setenv SIMPRIMS $LCA/simprims
```

Also verify that your $MGC_LOCATION_MAP file contains the following entries.

```
$LCA

(blank)

$SIMPRIMS

(blank)
```

Synopsys Interface Environment Variables

To use LogiBLOX with Synopsys, add the following entries to the .synopsys_vss. setup file, located in the working directory.

```
logiblox:

$XILINX/synopsys/libraries/sim/logiblox/lib
```

Viewlogic Interface Environment Variables

To use LogiBLOX with Viewlogic, add the following path to the WDIR environment variable.

```
${XILINX}/viewlog/data/logiblox/standard\
```

The following is an example.

```
setenv WDIR ${XILINX}/viewlog/data/logiblox/standard:<existing-WDIR>
```

Verify that the following two libraries have been added to the search order in the local viewdraw.ini file right before the “builtin” and “xbuiltin” libraries. Modify the file with the following entries.

```
DIR [r]/xilinx_path/viewlog/data/logiblox (logiblox)

DIR [m]/xilinx_path/viewlog/data/simprims (simprims)
```

Setting Up LogiBLOX on a PC

This section describes the issuing commands and the file modifications required to set up your environment when using a third-party schematic design tool on a PC. You must set up the Xilinx environment and interface environment described in the Software Installation section of the *Alliance 2.1i Release Notes and Installation Guide*, and in the *LogiBLOX Guide*.

Viewlogic Environment Variables for PCs

To use LogiBLOX with Workview Office, run the following command in an MS-DOS session.

```
custmenu xilinx_path\viewlog\data\viewblox.txt
```

Verify the following two libraries are included in the search order in the Viewlogic Project Manager right before the “builtin” and “xbuiltin” libraries.

```
[r] <xilinx_path>\viewlog\data\logiblox (logiblox)
[m] <xilinx_path\viewlog\data\simprims (simprims)
```

Starting LogiBLOX

LogiBLOX can be started in one of three ways.

- From a third-party vendor tool by selecting the menu item that lists LogiBLOX as a menu choice
- From a DOS or UNIX command line by entering.

  `lbgui`
- From the LogiBLOX icon in the Xilinx program group (PC only)

Using LogiBLOX for Schematic Design

LogiBLOX modules can be created for use in schematic designs using third-party design tools. First, the module must be created. The module can then be added to the schematic like any other library component with the aid of the LogiBLOX GUI.

Creating a LogiBLOX Module

To create a LogiBLOX module, follow these steps.

1. From ViewDraw or Mentor Graphics, select the appropriate menu choice in your design tool to start LogiBLOX. The LogiBLOX Module Selector dialog box appears.
   - In Mentor Graphics, from Pld_da select the following.

     `Library` → **`Xilinx Library`** → **`LogiBLOX`**

     An intermediate dialog window named create / modify / instantiate LogiBLOX symbol appears before the LogiBLOX GUI. This window replaces the LogiBLOX Setup dialog box.
   - In Viewlogic on a workstation, select the following from the ViewDraw window.

     `Add` → **`LogiBLOX`**
   - In Viewlogic on a PC, select.

     `Tools` → **`Add LogiBLOX`**

The LogiBLOX Module Selector dialog window is displayed. If a logiblox.ini file is not found, the LogiBLOX Setup dialog box displays before the Module Selector dialog box.

2. Select a base module type (for example, Counter, Memory).
3. Customize the module by selecting pins and specifying attributes.
4. Click **OK**.

LogiBLOX generates a schematic symbol and a simulation model for the module you have selected.

Note: You can add existing LogiBLOX components from the project library.

Design Simulation

You can functionally simulate your design at any time.

At this point the design is ready to be processed for both simulation and implementation. Because LogiBLOX creates a component with a VHDL or EDIF simulation model describing its behavior, the simulation and implementation flow for a design containing LogiBLOX components is no different than for a design which does not contain LogiBLOX components. Once created, the LogiBLOX components can be used repeatedly in any design.

Copying Modules

If you copy a module within your schematic or add repeated instances, the original module and all of its copies share the same .mod file and simulation model. Subsequent modifications to any one of these modules changes all copies of that module. If you copy a module from another design, such as by copying an entire hierarchical module, you must invoke the LogiBLOX program and cause it to regenerate the module and re-create the simulation model for that module. Alternatively, if your design includes several copied modules, you can copy the raw HDL files into the new project directory and re-analyze them in the new environment.

Using LogiBLOX for HDL Synthesis Design

LogiBLOX modules can be instantiated in HDL designs to address special features, such as distributed memory (XC4000E and XC4000EX), special I/O configurations, and other advanced silicon features that cannot be inferred by the HDL synthesizer.

The LogiBLOX program creates a simulation netlist (VHDL, EDIF or Verilog), an implementation netlist file (.ngc), and a template file containing a VHDL (.vhi) or Verilog (.vei) component instantiation.

Instantiating a LogiBLOX Module

To instantiate a LogiBLOX module, proceed with the following steps.

1. Start LogiBLOX from the command line, or click on the Logi-BLOX icon. See the “Starting LogiBLOX” section.
2. Select Setup on the Module Selector dialog box. The Setup dialog box appears.
3. The Setup dialog window displays initially if a logiblox.ini file is not found in the home directory.
4. Select Options. The Options selections appear.
5. Select the Simulation model you require (VHDL, EDIF, or Verilog).
6. Click OK. The Setup dialog box disappears.
7. Create the module you want in the LogiBLOX Module Selector dialog box.
8. Click OK.
9. Instantiate the module in the top level.

   With a text editor, cut and paste the contents of the VHDL (*.vhi*) or verilog (.vei) design file to the top level design. Then, specify the design names in the component instantiation section.

Analyzing a LogiBLOX Module

Before starting behavioral simulation on an instantiated LogiBLOX module, the LogiBLOX library has to be analyzed. The following three sections list the commands that can be used in Mentor, Synopsys, and Viewlogic to analyze the library.

Mentor QuickHDL

Enter the following series of commands from your workstation command line to analyze the LogiBLOX libraries.

`$XILINX/mentor/data/vhdl/compile_vhdl_libs.sh (VHDL)`

`$XILINX/mentor/data/verilog/compile_verilog_libs.sh (Verilog)`

See the accompanying README files in the same directories for more information on these scripts.

Synopsys VSS

Enter the following series of commands from your workstation command line to analyze the LogiBLOX libraries.

`$XILINX/synopsys/libraries/sim/src/logiblox/analyze.csh`

The script analyzes the model and places the output files in the $XILINX/synopsys/libraries/sim/lib/logiblox directory.

Viewlogic Vantage

Enter the following command from your workstation/PC command line to analyze the LogiBLOX libraries.

`vaninit` *parent_directory*

This is a script provided by Xilinx. The new logiblox.lib Vantage library directory will be created under the specified parent_directory. You do not need to re-analyze the LogiBLOX library for every new project.

MTI Modelsim

Enter the following command from your workstation/PC command line to analyze the LogiBLOX libraries.

VHDL Designs

vlib */destination/path* **/simprim**

vmap simprim */destination/path* **/simprim**

vcom -work simprim $XILINX/vhdl/src/simprims/ simprim_Vpackage.vhd

vcom -work simprim $XILINX/vhdl/src/simprims/ simprim_Vcomponents.vhd

vcom -work simprim $XILINX/vhdl/src/simprims/ simprim_VITAL.vhd

Verilog Designs

vlib */destination/path* **/simprim**

vmap simprim */destination/path* **/simprim**

vlog -work simprim $XILINX/verilog/src/*.vmd

LogiBLOX Modules

LogiBLOX has many different modules that you can use in a schematic or HDL synthesis design. The following is a list of the LogiBLOX modules.

- Accumulator
- Adder/Subtracter
- Clock Divider
- Comparator
- Constant
- Counter
- Data Register
- Decoder
- Input/Output
- Memory
- Multiplexer

- Pad
- Shift Register
- Simple Gates
- Tristate

Appendix F

Instantiated Components

This appendix lists the components most frequently instantiated in synthesis designs. The function of each component is briefly described and the pin names are supplied, along with a listing of the Xilinx product families involved. For a complete list of components, see the online version of the *Libraries Guide* and *Xilinx Synopsys Interface Guide*, or your synthesis tool documentation. This chapter contains the following sections.

- "STARTUP Component"
- "STARTBUF Component"
- "BSCAN Component"
- "READBACK Component"
- "RAM and ROM"
- "Global Buffers"
- "Fast Output Primitives"
- "IOB Components"
- "Clock Delay Components"

STARTUP Component

The STARTUP component is used to access the global set/reset and global tristate signals. STARTUP can also be used to access the startup sequence clock. For information on the start-up sequence and the associated signals, see *The Programmable Logic Data Book* and the Xilinx *Libraries Guide*.

The STARTUP component cannot be simulated. For VHDL designs, use components in the following table.

Table F-1 STARTUP Library Component

Name	Family	Description	Outputs	Inputs
STARTUP	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, 5200[a]	Used to connect Global Set/Reset, global tristate control, and user configuration clock.	Q2, Q3, Q1Q4, DONEIN	GSR, GTS, CLK

a. For 5200, GSR pin is GR.

STARTBUF Component

The STARTBUF component allows you to functionally simulate the STARTUP component. As with STARTUP, a STARTBUF component instantiated in your design specifies to the implementation tools to use GSR. Using the STARTBUF component in VHDL designs is the preferred method for using GSR/GR.

Table F-2 STARTBUF Library Component

Name	Family	Description	Outputs	Inputs
STARTBUF	4000E/L, 4000EX, 4000XL, 4000XV, 5200	Used to connect Global Set/Reset, global tristate control, and user configuration clock.	GSROUT, GTSOUT,Q2OUT, Q3OUT, Q1Q4OUT, DONEINOUT	GSRIN, GTSIN, CLKIN

BSCAN Component

To use the boundary-scan (BSCAN) circuitry in a Xilinx FPGA, the BSCAN component must be present in the input design. The TDI, TDO, TMS, and TCK components are typically used to access the reserved boundary-scan device pads for use with the BSCAN component but can be connected to user logic as well. For more information on the BSCAN component, the internal boundary-scan circuitry, and the directional properties of the four reserved boundary-scan pads, refer to *Programmable Logic Data Book* and the online version of the Xilinx *Libraries Guide.*

Table F-3 BSCAN Library Components

Name	Family	Description	Outputs	Inputs
BSCAN	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, 5200	Indicates that the boundary- scan logic should be enabled after the FPGA has been configured.	TDO, DRCK, IDLE, SEL1, SEL2	TDI, TMS, TCK, TDO1, TDO2
TDI	4000E/L, 4000EX, 4000XL, 4000XV, 5200	Connects to the BSCAN TDI input. Loads instructions and data on each low-to-high TCK transition.	I	—
TDO	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, 5200	Connects to the BSCAN TDO output. Provides the boundary-scan data on each low-to-high TCK transition.	—	O
TMS	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, 5200	Connects to the BSCAN TMS input. It determines which boundary scan is performed.	I	—
TCK	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, 5200	Connects to the BSCAN TCK input. Shifts the serial data and instructions into and out of the boundary-scan data registers.	I	—

Note: The 5200 has three additional output pins: Reset, Update, and Shift.

READBACK Component

To use the dedicated readback logic in a Xilinx FPGA the READBACK component must be inserted in the input design. The MD0, MD1, and MD2 components are typically used to access the mode pins for use with the readback logic, but can be connected to user logic as well. For more information on the READBACK component, the internal readback logic, and the directional properties of the three reserved mode pins, see the *Programmable Logic Data Book* and the online manual *Libraries Guide*.

Table F-4 Readback Library Components

Name	Family	Description	Outputs	Inputs
READBACK	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, 5200	Accesses the bitstream readback function. A low-to-high transition on the TRIG input initiates the readback process.	DATA, RIP	CLK, TRIG
MD0	4000E/L, 4000EX, 4000XL, 4000XV, 5200	Connects to the Mode 0 (M0) input pin, which is used to determine the configuration mode.	I	—
MD1	4000E/L, 4000EX, 4000XL, 4000XV, 5200	Connects to the Mode 1 (M1) input pin, which is used to determine the configuration mode.	—	O
MD2	4000E/L, 4000EX, 4000XL, 4000XV, 5200	Connects to the Mode 2 (M2) input pin, which is used to determine the configuration mode.	I	—

RAM and ROM

Some of the most frequently instantiated library components are the RAM and ROM primitives. Because most synthesis tools are unable to infer RAM or ROM components from the source HDL, the primitives must be used to build up more complex structures. The following list of RAM and ROM components (Table G-4) is a complete list of the primitives available in the Xilinx library. For more information on the components, see the *Programmable Logic Data Book* and the online manual *Libraries Guide*.

Table F-5 RAM and ROM Library Components

Name	Family	Description	Outputs	Inputs
RAM16X1	4000E/L, 4000EX, 4000XL, 4000XV	A 16-word by 1-bit static read-write random-access memory component.	O	D, A3, A2, A1, A0, WE
RAM16X1D	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, Virtex	A 16-word by 1-bit dual port random access memory with synchronous write capability and asynchronous read capability.	SPO, DPO	D, A3, A2, A1, A0, DPRA3, DPRA2, DPRA1, DPRA0, WE, WCLK
RAM16X1S	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, Virtex	A 16-word by 1-bit static random access memory with synchronous write capability and asynchronous read capability.	O	D, A3, A2, A1, A0, WE, WCLK
RAM32X1	4000E/L, 4000EX, 4000XL, 4000XV	A 32-word by 1-bit static read-write random access memory.	O	D, A0, A1, A2, A3, A4, WE
RAM32X1S	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, Virtex	A 32-word by 1-bit static random access memory with synchronous write capability and asynchronous read capability.	O	D, A4, A3, A2, A1, A0, WE, WCLK

Table F-5 RAM and ROM Library Components

Name	Family	Description	Outputs	Inputs
ROM16X1	4000E/L, 4000EX, 4000XL, 4000XV, Spartan	A 16-word by 1-bit read-only memory component.	O	A3, A2, A1, A0
ROM32X1	4000E/L, 4000EX, 4000XL, 4000XV, Spartan	A 32-word by 1-bit read-only memory component.	O	A4, A3, A2, A1, A0

Global Buffers

Each XC4000EX and XC4000XL device has 16 available global buffers: 8 BUFGLSs and 8 BUFEs. For some designs it may be necessary to use the exact buffer desired to ensure appropriate clock distribution delay. For most designs, the BUFG, BUFGS, and BUFGP components can be inferred or instantiated, thus allowing the Alliance Series Design Implementation Tools to make an appropriate physical buffer allocation. For more information on the components, see the *Programmable Logic Data Book.*

Table F-6 Global Buffers Library Components

Name	Family	Description	Outputs	Inputs
BUFG	3000A, 3100A, 4000E/L, 4000EX, 4000XL, 4000XV, Spartan, 5200, Virtex	An architecture-independent global buffer, distributes high fan-out clock signals throughout a PLD device.	O	I
BUFGP	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, Virtex	A primary global buffer, distributes high fan-out clock, or control signals throughout PLD devices.	O	I
BUFGS	4000E/L, 4000EX, 4000XL, 4000XV, Spartan	A secondary global buffer, distributes high fan-out clock, or control signals throughout a PLD device.	O	I
BUFGLS	4000EX, 4000XL, 4000XV	Global Low-Skew buffer. BUFGLS components can drive all flip-flop clock pins.	O	I
BUFGE	4000EX, 4000XL, 4000XV	Global Early buffer. XC4000EX devices have eight total, two in each corner. BUFGE components can drive all clock pins in their corner of the device.	O	I

Fast Output Primitives

One of the features added to the XC4000EX and XC4000XL architectures is the fast output MUX. There is one fast output MUX located in each IOB which can be used to multiplex between two signals on a single device pad or can be used to implement any two input logic function. Each component can have zero, one, or two inverted inputs. Because the output MUX is located in the IOB, it must be connected to the input pin of either an OBUF or an OBUT. For more information on the output primitives, see the *Programmable Logic Data Book.* For information on how to instantiate output MUXs with inverted inputs, see the *Synopsys (XSI) Interface/Tutorial Guide.*

Table F-7 Fast Output Primitives

Name	Family	Description	Outputs	Inputs
OAND2	4000EX, 4000XL	2-input AND gate that is implemented in the output multiplexer of the XC4000EX/XL IOB	O	F, I0
ONAND2	4000EX, 4000XL	2-input NAND gate that is implemented in the output multiplexer of the XC4000EX/XL IOB	O	F, I0
OOR2	4000EX, 4000XL	2-input OR gate that is implemented in the output multiplexer of the XC4000EX/XL IOB.	O	F, I0
ONOR2	4000EX, 4000XL	2-input NOR gate that is implemented in the output multiplexer of the XC4000EX/XL IOB.	O	F, I0
OXOR2	4000EX, 4000XL	2-input exclusive OR gate that is implemented in the output multiplexer of the XC4000EX/XL IOB.	O	F, I0
OXNOR2	4000EX, 4000XL	2-input exclusive NOR gate that is implemented in the output multiplexer of the XC4000EX/XL IOB.	O	F, I0
OMUX2	4000EX, 4000XL	2 x 1 MUX implemented in the output multiplexer of the XC4000EX/XL IOB.	O	D0, D1, S0

IOB Components

Depending on the synthesis vendor being used, some IOB components must be instantiated directly in the input design. Most synthesis tools support IOB D-type flip-flop inferences, but may not yet support IOB D-type flip-flop inference with clock enables. Because there are many slew rates and delay types available, there are many derivatives of the primitives shown. For a complete list of the IOB primitives, see the *Synopsys (XSI) Interface/Tutorial Guide.*

Table F-8 IOB Components

Name	Family	Description	Outputs	Inputs
IBUF	3000A, 3100A, 4000E/L, 4000EX, 4000XL, 4000XV, Spartan, 5200, Virtex	Single input buffers. An IBUF isolates the internal circuit from the signals coming into a chip.	O	I
OBUF	3000A, 3100A, 4000E/L, 4000EX, 4000XL, 4000XV, Spartan, 5200, Virtex	Single output buffers. An OBUF isolates the internal circuit and provides drive current for signals leaving a chip.	O	I
OBUFT	3000A, 3100A, 4000E/L, 4000EX, 4000XL, 4000XV, Spartan, 5200, Virtex	Single tristate output buffer with active-low output enable. (tristate High)	O	I,T

Table F-8 IOB Components

Name	Family	Description	Outputs	Inputs
IFD	3000A, 3100A, 4000E/L, 4000EX, 4000XL, 4000XV, Spartan, 5200, Virtex	Single input D flip-flop.	Q	D, C
OFD	3000A, 3100A, 4000E/L, 4000EX, 4000XL, 4000XV, Spartan, 5200, Virtex	Single output D flip-flop.	Q	D, C
OFDT	3000A, 3100A, 4000E/L, 4000EX 4000XL, 4000XV, Spartan, 5200, Virtex	Single D flip-flop with active-high tristate active-low output enable buffers.	O	D, C,T
IFDX	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, Virtex	Single input D flip-flop with clock enable.	Q	D, CE, C

Table F-8 IOB Components

Name	Family	Description	Outputs	Inputs
OFDX	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, Virtex	Single output D flip-flop with clock enable	Q	D, C, CE
OFDTX	4000E/L, 4000EX, 4000XL, 4000XV, Spartan	Single D flip-flop with active-high tristate and active-low output enable buffers.	O	D, C, CE, T
ILD_1	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, 5200, Virtex	Transparent input data latch with inverted gate. (Transparent High).	Q	D, G

Clock Delay Components

These components are delay locked loops that are used to eliminate the clock delay inside the device. The delay locked loop is a digital variation of the analog phase locked loop.

Table F-9 Clock Delay Component

Name	Family	Description	Outputs	Inputs
CLKDLL	Virtex	Clock delay locked loop used to minimize clock skew.	CLK0, CLK90, CLK180, CLK270, CLS2X, CLKDV, LOCKED	CLKIN, CLKFB, RST
CLKDLLHF	Virtex	High frequency clock delay locked loop used to minimize clock skew.	CLK0, CLK180, CLKDV, LOCKED	CLKIN, CLKFB, RST

Appendix G

Alliance Constraints

This appendix briefly describes the Constraints Editor tool. It also describes how to use constraints with LogiBLOX RAM/ROM and Synopsys tools. For a complete listing of all supported constraints, refer to the *Libraries Guide*. For a more complete description of timing and layout constraints, block delay symbols, and supported constraints, refer to the *Constraints Editor Guide*. This appendix includes the following sections.

- “Entering Design Constraints”
- “Translating and Merging Logical Designs”
- “Constraining LogiBLOX RAM/ROM with Synopsys”

Entering Design Constraints

The Xilinx tools allow you to control the implementation of your design by entering constraints. You can enter constraints during the design and implementation phases of the design flow. The following figure illustrates where constraints entry fits in the overall design flow.

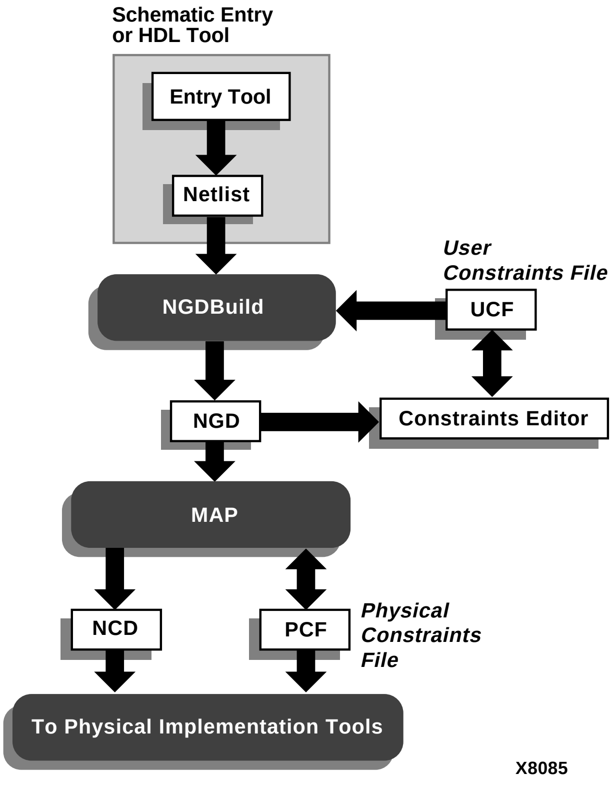

Schematic Entry or HDL Tool
Entry Tool
Netlist
User Constraints File
UCF
NGDBuild
NGD
Constraints Editor
MAP
NCD
PCF
Physical Constraints File
To Physical Implementation Tools
X8085

Figure G-1 Entering Design Constraints

During the design phase, you can enter constraints as follows.

- Add constraints to your schematic
- Add constraints to your design in your synthesis tool
- Enter constraints in the Xilinx Constraints Editor

You can apply location and timing constraints to your design. Use location constraints to control the mapping and positioning of the logic elements in the target device. The most common location constraints are pad constraints. Pad constraints are used to lock the pins of your design to specific I/O locations so that the pin placement is consistent from revision to revision. Use timing constraints to specify how fast a path must be to meet your speed requirements.

You can use timing constraints for the placement and routing of your design.

Constraints entered directly in your input design are known as design constraints, and are eventually placed in your design netlist. If you want the constraints separated from your input design files, or if you want to modify your constraints without re-synthesizing your design, you can create a User Constraints File (UCF) in the Constraints Editor. This file is read by NGDBuild during the translation of your design, and is combined with an EDIF or XNF netlist into an NGD file. If a UCF file exists with the same name as the top-level netlist, it is automatically read. Otherwise, you must specify a file name for User Constraints in the Options dialog box.

Adding Constraints with the Constraints Editor

The Constraints Editor is a new graphical tool in the Xilinx Development System that allows you to enter timing constraints and pin location constraints. You can enter constraints in the graphical interface without understanding UCF file syntax. The Constraints Editor passes these constraints to the implementation tools through a UCF file.

The Constraints Editor accepts the following input files.

- A valid NGD file, which is a Xilinx logical design database file. This file serves as input to the Map program, which generates the physical design database (NCD).
- A corresponding UCF (User Constraints File), which contains logical constraints.

By default, when the NGD file is opened, an existing UCF file with the same base name as the NGD file is used. Alternatively, you can specify the name of the UCF file.

The Constraints Editor writes out a valid UCF file and a valid NGD file.These files are processed by the Map program, which generates a PCF (Physical Constraints File).

Using the Global Tab

The main window of the Constraints Editor GUI is called the Global Tab. Following are the main options of the Pad to Pad button, which is an integral part of this window.

- Timespec Name: you can enter your timespecs for pad to pad constraints in this field.
- Time: enter your specified value in this field.
- Units: you can enter your units in measurements of ps, ns, us, and ms (picoseconds, nanoseconds, microseconds, and milliseconds, respectively.)

For more information on using the Constraints Editor GUI and entering constraints, refer to the *Constraints Editor Guide.*

The Constraints Editor adds timing constraints to the UCF (User Constraints File) which is used translate a netlist into your design.

Note: You must re-translate your design after running the Constraints Editor.

Translating and Merging Logical Designs

The process of implementing a design with the Xilinx tools starts by constructing a logical design file (NGD) that represents the design created by the NGDBuild application. The NGD file contains all of the design's logic structures (gates) and constraints. NGDBuild controls the translation and merging of all of the related logic design files. All design files are translated from industry standard netlists to intermediate NGO files by XNF2NGD or EDIF2NGD netlist translation programs.

Constraining LogiBLOX RAM/ROM with Synopsys

In the XSI HDL methodology, whenever large blocks of RAM/ROM are needed, LogiBLOX RAM/ROM modules are instantiated in the HDL code. With LogiBLOX RAM/ROM modules instantiated in the HDL code, timing and/or placement constraints on these RAM/ROM modules, and the RAM/ROM primitives that comprise these modules, can be specified in a .ucf file. To create timing and/or placement constraints for RAM/ROM LogiBLOX modules, knowledge of how many primitives will be used and how the primitives, and/or how the RAM/ROM LogiBLOX modules are named is needed.

Estimating the Number of Primitives Used

When a RAM/ROM is specified with LogiBLOX, the RAM/ROM depth and width are specified. If the RAM/ROM depth is divisible by 32, then 32x1 primitives are used. If the RAM/ROM depth is not divisible by 32, then 16x1 primitives are used instead. In the case of dual-port RAMs, 16x1 primitives are always used. Based on whether 32x1 or 16x1 primitives are used, the number of RAM/ROM can be calculated.

For example, if a RAM48x4 was required for a design, RAM16x1 primitives would be used. Based on the width, there would be four banks of RAM16x1s. Based on the depth, each bank would have three RAM16x1s.

Naming RAM Primitives

Using the example of a RAM48x4, the RAM primitives inside the LogiBLOX would be named as follows.

```
MEM0_0  MEM1_0  MEM2_0  MEM3_0
MEM0_1  MEM1_1  MEM2_1  MEM3_1
MEM0_2  MEM1_2  MEM2_2  MEM3_2
```

Each primitive in a LogiBLOX RAM/ROM module has a instance name of MEMx_y, where y represents the primitive position in the bank of memory, and where x represents the bit position of the RAM/ROM output.

For the next two items, refer to the Verilog/VHDL examples included at the end of this section. The Verilog/VHDL example instantiates a RAM32x2S, which is in the bottom of the hierarchy. The RAM32x2S was made with LogiBLOX. The next two items are written within the context of the Verilog examples, but also apply to the VHDL examples as well. Note, the runscripts included were designed for FPGA Compiler. If you want to use Design Compiler, remove the replace_fpga step.

Referencing a LogiBLOX Module

LogiBLOX RAM/ROM modules in the FPGA/Design Compiler flow are constrained via a .ucf file. LogiBLOX RAM/ROM modules instantiated in the HDL code can be referenced by the full-hierarchical instance name. If a LogiBLOX RAM/ROM module is at the top-level of the HDL code, then the instance name of the LogiBLOX RAM/ROM module is just the instantiated instance name.

In the case of a LogiBLOX RAM/ROM, which is instantiated within the hierarchy of the design, the instance name of the LogiBLOX RAM/ROM module is the concatenation of all instances which contain the LogiBLOX RAM/ROM. For FPGA/Design Compiler, the concatenated instance names are separated by a "/¨. In the example, the RAM32X1S is named *memory.* The module *memory* is instantiated in Verilog module *inside* with an instance name U0. The module *inside* is instantiated in the top-level module test. Therefore, the RAM32X1S can be referenced in a .ucf file as U0/U0. For example, to attach a TNM to this block of RAM, the following line could be used in the .ucf file.

```
INST U0/U0 TNM=block1;
```

Since U0/U0 is composed of two primitives, a timegroup called block1 would be created; block1 TNM could be used throughout the .ucf file as a Timespec end/start point, and/or or U0/U0 could have a LOC area constraint applied to it. If the RAM32X1S has been instantiated in the top-level file, and the instance name used in the instantiation was U0, then this block of RAM could just be referenced by U0.

If FPGA Express is the tool being used, then the concatenated instance names are separated by a "_" instead.

```
INST U0_U0 TNM=block1;
```

Referencing LogiBLOX Module Primitives

Sometimes its necessary to apply constraints to the primitives that compose the LogiBLOX RAM/ROM module. For example, if you choose a floorplanning strategy to implement your design, it may be necessary to apply LOC constraints to one or more primitives inside a LogiBLOX RAM/ROM module.

Consider the previous RAM32x2S example, suppose that the each of the RAM primitives had to be constrained to a particular CLB location. Based on the rules for determining the MEMx_y instance names, using the previous example, each of RAM primitives could be referenced by concatenating the full-hierarchical name to each of the MEMx_y names. The RAM32x2S created by LogiBLOX would have primitives named MEM0_0 and MEM1_0. So, for FPGA/Design

Compiler, CLB constraints in a .ucf file for each of these two items would be:

```
INST U0/U0/MEM0_0 LOC=CLB_R10C10;
INST U0/U0/MEM0_1 LOC=CLB_R11C11;
```

For FPGA Express, the CLB constraints would be:

```
INST U0_U0/MEM0_0 LOC=CLB_R10C10;
INST U0_U0/MEM0_1 LOC=CLB_R11C11;
```

FPGA/Design Compiler and Express Verilog Examples

This section includes FPGA / Design Compiler and Express Verilog Examples.

Test.v Example

```
module test(DATA,DATAOUT,ADDR,C,ENB);

input [1:0] DATA;
output [1:0] DATAOUT;
input [4:0] ADDR;
input C;
input ENB;
wire [1:0] dataoutreg;
reg [1:0] datareg;
reg [1:0] DATAOUT;
reg [4:0] addrreg;

inside U0
(.MDATA(datareg),.MDATAOUT(dataoutreg),.MADDR(addrreg
),.C(C),.WE(ENB));

always@(posedge C) datareg = DATA;

always@(posedge C) DATAOUT = dataoutreg;

always@(posedge C) addrreg = ADDR; endmodule
```

Inside.v Example

```
module inside(MDATA,MDATAOUT,MADDR,C,WE);

input [1:0] MDATA;
output [10] MDATAOUT;
input [4:0] MADDR;
input C;
input WE;

memory U0 ( .A(MADDR), .DO(MDATAOUT), .DI(MDATA),
.WR_EN(WE), .WR_CLK(C));

endmodule
```

Memory.v Example (FPGA/Design compiler only)

```
module memory(A, DO, DI, WR_EN, WR_CLK);

input [4:0] A;
output [1:0] DO;
input [1:0] DI;
input WR_EN;
input WR_CLK;
endmodule
```

Runscript Example (FPGA/Design compiler only)

```
TOP=test part = “4028expg299-3”
read -f verilog “guts.v”
read -f verilog “inside.v”
read -f verilog “test.v”
current_design TOP
remove_constraint -all
set_port_is_pad “*”
insert_pads
compile
write -format db -hierarchy -output TOP +
“_compiled.db”
replace_fpga
set_attribute TOP “part” -type string part
write -format db -hierarchy -output TOP + “.db”
ungroup -all -flatten
write_script > TOP + “.dc” sh dc2ncf test.dc
remove_design guts
write -f xnf -h -o TOP + “.sxnf
```

Test.ucf Example (FPGA/Design compiler only)

```
INST “U0/U0” TNM = usermem;
TIMESPEC TS_6= FROM : FFS :TO: usermem: 50;
INST “U0/U0/mem0_0” LOC=CLB_R7C2;
```

Test.ucf Example (FPGA Express only)

```
INST “U0_U0” TNM = usermem;
TIMESPEC TS_6= FROM : FFS :TO: usermem: 50;
INST “U0_U0/mem0_0” LOC=CLB_R7C2;
```

FPGA/Design Compiler and Express VHDL Examples

This section includes FPGA / Design Compiler and Express VHDL Examples.

Test.vhd Example

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity test is
port( DATA: in STD_LOGIC_VECTOR(1 downto 0);
        DATAOUT: out STD_LOGIC_VECTOR(1 downto 0);
        ADDR: in STD_LOGIC_VECTOR(4 downto 0);
        C, ENB: in STD_LOGIC);
end test;

architecture details of test is
signal dataoutreg,datareg: STD_LOGIC_VECTOR(1 downto 0);
signal addrreg: STD_LOGIC_VECTOR(4 downto 0);

component inside
        port(MDATA: in STD_LOGIC_VECTOR(1 downto 0);
                MDATAOUT: out STD_LOGIC_VECTOR(1 downto 0);
                MADDR: in STD_LOGIC_VECTOR(4 downto 0);
                C,WE: in STD_LOGIC);
end component;

begin
        U0: inside port
map(MDATA=>datareg.,MDATAOUT=>dataoutreg.,MADDR=>addrreg,C=>C,WE=>ENB);

        process( C )
             begin
                  if(Cevent and C=1) then
                       datareg <= DATA;
                  end if;
             end process;

        process( C )
             begin
```

```
                  if(Cevent and C=1) then
                       DATAOUT <= dataoutreg;
                  end if;
             end process;

        process( C )
             begin
                  if(Cevent and C=1) then
                       addrreg <= ADDR;
                  end if;
             end process;

end details;
```

Inside.vhd Example

```
entity inside is
   port(
        MDATA: in STD_LOGIC_VECTOR(1 downto 0);
        MDATAOUT: out STD_LOGIC_VECTOR(1 downto 0);
        MADDR: in STD_LOGIC_VECTOR(4 downto 0);
        C,WE: in STD_LOGIC);
   end inside;

architecture details of inside is component memory
   port(
        A: in STD_LOGIC_VECTOR(4 downto 0);
        DO: out STD_LOGIC_VECTOR(1 downto 0);
        DI: in STD_LOGIC_VECTOR(1 downto 0);
        WR_EN,WR_CLK: in STD_LOGIC);
   end component;

begin
        U0: memory port map(A=>MADDR,DO=>MDATAOUT,
                DI=>MDATA,WR_EN=>WE,WR_CLK=>C);
end details;
```

Runscript Example (FPGA/Design compiler only)

```
TOP=test part = “4028expg299-3”
analyze -f vhdl “guts.vhd”
analyze -f vhdl “inside.vhd”
analyze -f vhdl “test.vhd”
elaborate TOP
current_design TOP
remove_constraint -all
set_port_is_pad “*”
insert_pads
```

```
compile
write -format db -hierarchy -output TOP +
“_compiled.db”
replace_fpga
set_attribute TOP “part” -type string part
write -format db -hierarchy -output TOP + “.db”
ungroup -all -flatten
write_script > TOP + “.dc” sh dc2ncf test.dc
remove_design guts
write -f xnf -h -o TOP + “.sxnf”
```

Test.ucf Example (FPGA/Design compiler only)

```
INST “U0/U0” TNM = usermem;
TIMESPEC TS_6= FROM : FFS :TO: usermem 50;
INST “U0/U0/mem0_0” LOC=CLB_R7C2;
```

Test.ucf Example (FPGA Express only)

```
INST “U0_U0” TNM = usermem;
TIMESPEC TS_6= FROM : FFS :TO: usermem: 50;
INST “U0_U0/mem0_0” LOC=CLB_R7C2;
```

Appendix H

Configuring Xprinter

This appendix provides detailed instructions on configuring a printer so you can print from the Xilinx application. The information in this appendix applies only to workstation applications. The following sections are in this Appendix.

- “Required Wind/U Files”
- “Configuring .WindU”
- “Solving Printing Problems”

Required Wind/U Files

You must have the following Wind/U files installed correctly to print from your application.

- PPD (PostScript Printer Description)

  PPD files provide Wind/U with model-specific information like paper tray configuration, supported paper sizes, available ROM fonts, and so forth.

- AFM (Adobe Font Metric) and TFM (Tagged Font Metric)

  AFM and TFM files provide font metric information for all fonts in PostScript and PCL5 printers, respectively.

- Additional files

  Various other files are copied to `$WUHOME/xprinter` and are required by Wind/U when printing. These include `xprinter.prolog` (PostScript prolog), `psstd.fonts` and `pclstd.fonts` (provide PostScript/PCL5 to X Window System font name mappings), and `rgb.txt`.

Make a complete, as-is copy of the directory `$WUHOME/xprinter` and include it in the installation for your product. None of the files from this directory require modifications in most environments.

Note: All of the files in $WUHOME/xprinter are the property of Bristol Technology and are licensed to you under the terms of the Wind/U license agreement. You can freely distribute these files as long as they are bundled with your application. Consult the Wind/U license agreement for further details.

Configuring .WindU

Once you have installed the required printer configuration files on your system, you must configure the .WindU file in your home directory (or SYS$LOGIN:WINDU.INI) for printing.

You can modify the .WindU file either by using a text editor or the Xprinter Printer Setup dialog box. Using the dialog box is recommended, because it reduces the risk of error. The instructions in this appendix describe how to edit the .WindU file, and then provide step-by-step instructions for performing the same task with the Printer Setup dialog box. These instructions are provided to help you configure your printer.

Printer Information and PPD Files

When you configure Wind/U to print, you need to know the following information for each printer you want to access.

- Name of the printer description (PPD) file
- The command used to send output to a printer

The Wind/U installation media includes the PPD files for most commonly used printers. To verify that the PPD file associated with your printer is included, look at the Printer Devices dialog (from the Printer Setup dialog, click Install and Add Printer). If your printer model is listed, Wind/U has a PPD file for your printer. If your printer model is not listed, contact your printer vendor to obtain the PPD file for your printer.

Unix Print Command

The command used to send output to a specific printer depends on the platform, the printer, and how the printer is connected to your system. For example, if a printer is connected directly to your system, the following might be valid print commands.

Unix	**`lp -d ps -t$XPDOCNAME`**
OpenVMS	**`PRINT /QUEUE=OPTRA`**

If your printer is connected to a different system on your network, your printer command will specify how to connect to that system. For example, if a printer is connected to the system bandit on your network, any of the following might be valid print commands

`rsh bandit lp -d ps -t$XPDOCNAME`

Note: In these examples, $XPDOCNAME represents the name of the output file sent to the printer with the specified command. If you use a multi-word file name, such as *a print file*, you must enclose the $XPDOCNAME in quotation marks as follows. You must escape the quotation marks in the remote command, because `rsh` strips them out if you do not.

The following table lists the commands needed to configure your remote or local printer.

Local Printer	**`lp -d ps -t”$XPDOCNAME”`**
Remote Printer	**`rsh bandit lp -d ps -t\”$XPDOCNAME\”`**

Configuring Wind/U for Printing

Once you know the name of the PPD file and the print command for each printer you want to direct output to, you can configure Wind/U to recognize those printers. To configure Wind/U to recognize a printer, you must do the following

1. Define a port, which is an alias for the print command.
2. Associate the port with the printer’s PPD file.
3. Specify a default printer.
4. Set printer options.

Defining a Port

A printer port is an alias for the print command. It is defined in the Ports section of $HOME/.WindU and appears as part of the Printer Name in the Printer Setup dialog box. For example, the following is the first Printer Name in the Printer Setup dialog box before you make any changes to $HOME/.WindU.

```
AppleLaserWriter v23.0 PostScript on FILE:
```

In this Printer Name, FILE: is the port name. Port entries in the [ports] section have the following format.

port=print_command

This command sends output to the printer *port*. For example, if you have two printers: ORION and SIRIUS, your `[ports]` section may look like the following example.

```
[ports]
ORION=rsh bandit lp -d ps -t\"$XPDOCNAME\"
SIRIUS=rsh bandit lp -d ps -T pcl5 -t\"$XPDOCNAME\"
```

In this example, both printers are connected to the system *bandit*, so the print command is a remote shell command executed on *bandit*. ORION is a PostScript printer, so the command `lp -d ps` is executed on *bandit* to print to ORION. SIRIUS, however, is a PCL5 printer, so the print command executed on *bandit* to print to SIRIUS is `lp -d ps -T pcl5`.

If you have a printer connected to your local system, you need to add an entry for it. For the local printer, add an entry similar to the following.

```
[ports]
ORION=rsh bandit lp -d ps -t\"$XPDOCNAME\"
SIRIUS=rsh bandit lp -d ps -T pcl5 -t\"$XPDOCNAME\"
LOCAL=lp -d ps -t$XPDOCNAME
```

Your printer port can be any name, except FILE:, which is the only reserved port name. It causes HyperHelp to create a print file formatted specifically for the specified printer type.

You must create an entry in the [ports] section for every printer you want to print to.

To Define a New Port

To define a new port using the Printer Setup dialog box, perform the following steps.

1. To display the Ports dialog box, from the Printer Setup dialog box, click **`Install`**, **`Add Printer`**, and **`Define New Port`**.

2. Type the port definition in the Edit Port edit box.
3. Click **`Add/Replace`**.

   The new port is now included in the list of current port definitions.
4. Repeat steps 1–3 for each printer you want to print to.

Note: To create a printer port for each available printer queue on HP700 systems, click the **`Spooler`** button in the Ports dialog box. This command creates a default printer port for each available printer queue returned by the **`lpstat -a`** command.

To Modify an Existing Port

To modify an existing port using the Printer Setup dialog box, perform the following steps.

1. To display the Ports dialog box, from the Printer Setup dialog box, click **`Install`**, **`Add Printer`**, and **`Define New Port`**.
2. Select the port you want to modify and edit the port information in the Edit Port edit box.
3. Click **`Add/Replace`**.

   The modified port is now included in the list of current port definitions.

Matching a Printer Type to a Defined Port

After you define a port for each printer, specify the type of printer associated with each port. Device types are listed in the `[devices]` section of the `.WindU` file. Each entry in the `[devices]` section has the following format.

alias=PPD_file driver,port

Note: There must be a space between the **`PPD_file`** and **`driver`** and a comma between the **`driver`** and the **`port`**.

The following table describes each part of this entry.

Field	Description
alias	The *alias* is a descriptive name that identifies the printer. It can be anything. The *alias* is the name of the printer that appears in the Printer Setup dialog box, such as, `HP LaserJet 4L PostScript`).
PPD_file	The *PPD_file* is the name of the printer description (PPD) file used by the printer, without the `.PPD` extension.
driver	The *driver* is the type of driver the printer uses. Valid values are `PostScript`, `PCL4`, and `PCL5`.
port	The *port* is the printer port listed in the `[ports]` section of the .WindU file. (ORION, SIRIUS, and LOCAL appear in the example `[ports]` section.)

The following table lists the the ports that correspond to each printer type.

Port	Printer Type	Output Type
ORION	HP LaserJet 4LPostScript	PostScript
SIRIUS	HP LaserJet 4M PCL Cartridge	PCL
LOCAL	QMS-PS 2200 v52.3	PostScript

Following is an example procedure for configuring three printers.

1. Choose an alias for each printer.

   To easily identify the printer you want to use from the Printer Setup dialog box, use these aliases.

   - HP LaserJet PS
   - HP LaserJet PCL
   - QMS PS

Note: If you use the Printer Setup dialog box to associate ports and PPD files, you cannot specify a printer alias. You must choose an alias from the predefined list that appears in the Printer Devices list box in the Add Printer dialog box. The corresponding PPD file is already associated with the printer aliases in this list box.

2. Identify the PPD file associated with each of these printers. In this example, the `[devices]` section of the `.WindU` file appears as follows.

   ```
   [devices]
   HP LaserJet PS=HP3SI523 PostScript,ORION
   HP LaserJet PCL=HP4M PCL,SIRIUS
   QMS PS=Q2200523 PostScript,LOCAL
   ```

   After you add these entries to your `.WindU` file, the following printer choices are available from the Printer Setup dialog box.

   ```
   HP LaserJet PS on ORION
   HP LaserJet PCL on SIRIUS
   QMS PS on LOCAL
   ```

To Match a Printer Device to a Port

To match a printer device to a port using the Printer Setup dialog, perform the following steps.

1. To display the **`Add Printer`** dialog box, from the Printer Setup dialog box, click **`Install`** and **`Add Printer`**.
2. In the **`Printer Devices`** field, select the description that matches the printer you are installing.

   If no description matches your printer, contact your printer vendor for a printer description (PPD) file and install it in the `$WUHOME/xprinter/ppds` directory.
3. Select the desired port in the **`Current Port Definitions`** list box and click **`Add Selected`**.

   The new printer is now included in the list of currently installed printers.

To Remove an Installed Printer

To remove a printer device / port combination using the Printer Setup dialog box, perform the following steps.

1. To display the **`Printer Installation`** dialog box, from the Printer Setup dialog box, click **`Install`**.
2. In the **`Currently Installed Printers`** list box, select the printer you want to remove and click **`Remove Selected`**.

Specifying a Default Printer

After all available printers are configured, you can make one of them the default printer. To specify a default printer in the Printer Setup dialog box, add an entry in the following format to the `[windows]` section of the `.WindU` file.

```
[windows]
device=PPD_file,driver,port
```

Provide the same information that you used in the `[devices]` section. Only the format of the entry is different; there is a comma between the *PPD_file* and the *driver* instead of a space.

For example, if you want the default printer to be the printer at port ORION, your [windows] section appears as follows.

```
[windows]
device=HP4L,PostScript,ORION
```

The printing-related sections of your `.WindU` file look like the following.

```
[windows]
device=HP4L,PostScript,ORION

[ports]
ORION=rsh bandit "lp -d ps -t"
SIRUS=rsh bandit "lp -d ps -T pcl5"
LOCAL=lp -d ps

[devices]
HP LaserJet PS=HP4L PostScript,ORION
HP LaserJet PCL=HP4M PCL,SIRIUS
QMS PS=Q2200523 PostScript,LOCAL
```

Whenever you make and save a change with the Printer Setup dialog box, the changes are written to the `.WindU` file in your home directory.

In your default `.WindU` file, the `[windows]` entry appears as follows.

```
[windows]
device=NULL,PostScript,FILE:
```

Because no PPD file is listed (`NULL`), the default in the Printer Setup dialog box is to print generic PostScript to a file. You can specify the file name and change the type of output to PCL in the Printer Setup dialog box.

To Specify a Default Printer

To specify a default printer using the Printer Setup dialog box, do the following.

1. To display the **`Options`** dialog box, from the Printer Setup dialog, click **`Options`**.
2. From the **`Printer Name`** drop-down list, select the desired printer and click **`OK`**.
3. Click **`Save`** in the Printer Setup dialog box.

Setting Printer Options

Because printer options vary between printers, use the Printer Setup dialog box to set them. Xprinter reads the PPD file to identify the specific options available for each printer.

1. Display the Printer Setup Dialog box.
2. Set all fields to the desired values.

   The following table describes all printer setup fields.

Option	Description
Output Format	Specify whether to send output to a file or to a printer. If you choose Printer Specific, you can send output to any printer type/port combination configured in your `$HOME/.WindU` file. If the port is FILE: (as in this example), Xprinter creates an output file specifically for the specified printer type. If you choose Generic (File Only), print output is sent to an Encapsulated PostScript or generic PCL file.
Printer	Appears only if you select **`Output Format: Printer Specific`**. It specifies the name of the default printer to send print output to. Click the **`Options`** button to specify a different printer.

File Name	Appears only if you select **`Output Format: Generic (File Only)`**. Type the name of the print file to create. To pipe print output to a command, type a **`!`** character as the first character, then specify the command to pipe output to. For example, to pipe output to the `lp` command, enter the following: **`!lp -d ps`**.
EPSF PCL4 PCL5	Appears only if you select **`Output Format: Generic (File Only)`**. Click this button to display a list of output file types and select the desired type. Available types are EPSF (Encapsulated PostScript), PCL4, and PCL5.
Orientation	Specify portrait or landscape.
Scale	To increase the size of the output, specify a value greater than 1.00. To reduce the size, specify a value less than 1.00. For example, a value of 2.00 doubles the size of the output; a value of 0.50 reduces it by half.
Copies	Specify the number of copies to print.

3. To set additional options, such as selecting a new printer or changing the page size, click **`Options`**.
4. Set all options to the desired values.

   The following table describes all printer options.

Option	Description
Printer Name	Changes the Printer in the Setup dialog box. Click the down arrow to display a list of configured printers.
Resolution	Specify printer resolution. Values vary.
Page Size	Specify paper size. Values vary.
Paper tray	Specify tray where paper is located. Values vary.

5. Click **Save** to apply your changes and make them the new default values.

Sending Output to a File

The default `$HOME/.WindU` file contains many printer devices, including the following.

```
HP LaserJet 4L PostScript=HP4L PostScript,FILE:
HP LaserJet 4M PCL Cartridge PCL5=HP4M PCL,FILE:
```

In all of the default entries, the port is `FILE:`, which is the only reserved port name. If you specify `FILE:` as the *port*, Wind/U creates a print file instead of sending output to a printer. When you use a PPD file, you generate PostScript or PCL output that is specific to the printer. If you use Output Format: Generic (File Only), you generate generic Encapsulated PostScript or PCL output.

For example, the HP LaserJet 4L PostScript entry creates a PostScript file that includes the characteristics of the HP 4L PostScript printer. The HP LaserJet 4M PCL entry creates a PCL file that includes the characteristics of the HP LaserJet 4M PCL printer.

You can also print to a file instead of a printer by selecting the Output Format: Generic (file only) option in the Printer Setup dialog box, but doing so creates a generic EPS or PCL print file that does not take advantage of any special characteristics of your particular printer.

Solving Printing Problems

If you have problems printing, use the following hints.

- Start with just printing to a PostScript file. You can use PostScript previewers (on Sun's pageview), to see the file. Adding spooling and PCL support later is easy.
- Ensure that the $WUHOME/xprinter files are installed correctly.
- Ensure that you have .WindU in your home directory.
- Check that the `printing` sample application works with the configuration you are using with your application.
- Ensure that there is a PPD file for the printer you are using. Xprinter requires a PPD file that describes the attributes (paper size, resolution, color capabilities, paper trays, and so forth). for each printer device you want to use. Wind/U includes a number of PPD files for common printers; however, these do not represent all supported printers. If you have customer whose printer is not included in the PPD files supplied with Wind/U, try the following.
    - Contact the printer manufacturer for the PPD file.
    - Download the PPD file from the Adobe FTP site (ftp.adobe.com:/pub/adobe/PPDfiles).
    - Use the PPD file for a similar output device.

If you continue to have problems, submit an SPR to Bristol Technical Support. Be sure to include a copy of your printer output and your `.WindU` file.

Appendix I

Glossary

This appendix contains definitions and explanations for terms used in this manual.

aliases

Aliases, or signal groups, are useful for probing specific groups of nodes.

attribute

Attributes are instructions placed on symbols or nets in an FPGA schematic to indicate their placement, implementation, naming, direction, or other properties.

AutoRoute

AutoRoute automatically routes the objects you specify.

block

A group consisting of one or more logic functions.

component

A component is an instantiation or symbol reference from a library of logic elements that can be placed on a schematic.

constraint

Constraints are specifications for the implementation process. There are several categories of constraints: routing, timing, area, mapping, and placement constraints.

Using attributes, you can force the placement of logic (macros) in CLBs, the location of CLBs on the chip, and the maximum delay between flip-flops. PAR does not attempt to change the location of constrained logic.

CLBs are arranged in columns and rows on the FPGA device. The goal is to place logic in columns on the device to attain the best possible placement from the point of view of performance and space.

Constraints Editor

The Constraints Editor is a Graphical User Interface (GUI) that can be used to modify or delete existing constraints or to add new constraints to a design.

DC2NCF

DC2NCF (design constraints to netlist constraints file) translates a Synopsys DC file to a Netlist Constraints File (NCF). The DC file is a Synopsys setup file containing constraints for the design.

guided mapping

An existing NCD file is used to “guide” the current MAP run. The guide file may be used at any stage of implementation: unplaced or placed, unrouted or routed.

HDL

HDL (Hardware Description Language).

Implementation Tools

A set of tools that comprise the mainstream programs offered in the Xilinx design implementation tools. The tools are: NGDBuild, MAP, PAR, NGDAnno, TRCE, all the NGD2 translator tools, BitGen, PROMGen, and the FPGA Editor.

LCA file

An LCA file is a mapped file of a Xilinx design produced by an earlier release.

LCA2NCD

LCA2NCD converts an LCA file to an NCD file. The NCD file produced by LCA2NCD can be placed and routed, viewed in EPIC, analyzed for timing, and back-annotated.

LogiBLOX

Xilinx design tool for creating high-level modules such as counters, shift registers, and multiplexers.

locking

Lock placement applies a constraint to all placed components in your design. This option specifies that placed components cannot be unplaced, moved, or deleted.

Logic Block Editor

The Logic Block Editor allows you to edit the internal logic of a selected programmable component. Use the Edit Block command to start the logic block editor.

macro

A macro is a component made of nets and primitives, flip-flops or latches, that implements high-level functions, such as adders, subtractors, and dividers. Soft macros and RPMs are types of macros.

A macro can be unplaced, partially placed or fully placed, and it can also be unrouted, partially routed, or fully routed. See also "physical macro."

MCS file

An MCS file is an output from the PROMGen program in Intel's MCS-86 format.

MDF file

An MDF (MAP directive file) file is a file describing how logic was decomposed when the design was originally mapped. The MDF file is used for guided mapping using Xilinx Development System software.

MFP File

An MFP file is generated by the Floorplanner and controls the mapping and placement of logic in the design according to the floorplan created by the user.

MRP file

An MRP (mapping report) file is an output of the MAP run. It is an ASCII file containing information about the MAP run.

NCD file

An NCD (netlist circuit description) file is the output design file from the MAP program, LCA2NCD, PAR, or EPIC. It is a flat physical design database which may or may not be placed and routed

NCF file

An NCF (netlist constraints file) file is produced by a synthesis vendor toolset, or by the DC2NCF program. This file contains constraints specified within the toolset. EDIF2NGD and XNF2NGD reads the constraints in this file and adds the constraints to the output NGO file.

NGC File

Binary file containing the implementation of a module in the design. If an NGC file exists for a module, NGDBuild reads this file directly, without looking for a source EDIF or XNF netlist. In HDL design flows, LogiBLOX creates an NGC file to define each module.

NGDAnno

The NGDAnno program distributes delays, setup and hold time, and pulse widths found in the physical NCD design file back to the logical NGD file. NGDAnno merges mapping information from the NGM file, and timing information from the NCD file and puts all this data in the NGA file.

NGA file

An NGA (native generic annotated) file is an output from the NGDAnno run. An NGA file is subsequently input to the appropriate NGD2 translation program.

NGD2EDIF

NGD2EDIF is a program that produces an EDIF 2.1.0 netlist in terms of the Xilinx primitive set, allowing you to simulate pre- and post-route designs.

NGD2VER

NGD2VER is a program that translates your design into a Verilog HDL file containing a netlist description of the design in terms of Xilinx simulation primitives for simulation only.

NGD2VHDL

NGD2VHDL is a program that translates your design into a Vital 3 compliant VHDL file containing a netlist description of your design in terms of Xilinx simulation primitives for simulation only.

NGDBuild

The NGDBuild program performs all the steps necessary to read a netlist file in XNF or EDIF format and create and NGD file describing the logical design.

NGD file

An NGD (native generic database) file is an output from the NGDBuild run. An NGD file contains a logical description of the design expressed both in terms of the hierarchy used when the design was first created and in terms of lower-level Xilinx primitives to which the hierarchy resolves. The NGD file is the input to MAP.

NGM file

An NGM (native generic mapping) file is an output from the MAP run and contains mapping information for the design. The NGM file is an input file to the NGDAnno program.

PAR (Place and Route)

PAR is a program that takes an NCD file, places and routes the design, and outputs an NCD file. The NCD file produced by PAR can be used as a guide file for reiterative placement and routing. The NCD file can also be used by the bitstream generator, BitGen.

path delay

A path delay is the time it takes for a signal to propagate through a path.

PCF file

The PCF file is an output file of the MAP program. It is an ASCII file containing physical constraints created by the MAP program as well as physical constraints entered by you. You can edit the PCF file from within EPIC.

physical Design Rule Check (DRC)

Physical Design Rule Check (DRC) is a series of tests to discover logical and physical errors in the design. Physical DRC is applied from EPIC, BitGen, PAR, and Hardware Debugger. By default, results of the DRC are written into the current working directory.

physical macro

A physical macro is a logical function that has been created from components of a specific device family. Physical macros are stored in files with the extension.nmc. A physical macro is created when EPIC is in macro mode. See also “macro.”

pin

A pin can be a symbol pin or a package pin. A package pin is a physical connector on an integrated circuit package that carries signals into and out of an integrated circuit. A symbol pin, also referred to as an instance pin, is the connection point of an instance to a net.

pinwires

Pinwires are wires which are directly tied to the pin of a site (i.e. CLB, IOB, etc.)

route

The process of assigning logical nets to physical wire segments in the FPGA that interconnect logic cells.

route-through

A route that can pass through an occupied or an unoccupied CLB site is called a route-through. You can manually do a route-through in EPIC. Route-throughs provide you with routing resources that would otherwise be unavailable.

states

The values stored in the memory elements of a device (flip-flops, RAMs, CLB outputs, and IOBs) that represent the state of that device for a particular readback (time). To each state there corresponds a specific set of logical values.

TRCE

TRCE (Timing Reporter and Circuit Evaluator) “trace” is a program that will automatically perform a timing analysis on a design using available timing constraints. The input to TRCE is a mapped NCD file and, optionally, a PCF file. The output from TRCE is an ASCII timing report which indicates how well the timing constraints for your design have been met.

(Historical note: TRCE should not be confused with the UNIX trace command. The UNIX trace command is used to trace system calls and signals).

TWR file

A TWR (Timing Wizard Report) file is an output from the TRCE program. A TWR file contains a logical description of the design expressed both in terms of the hierarchy used when the design was first created and in terms of lower-level Xilinx primitives to which the hierarchy resolves.

wire

A wire is either: 1) a net or 2) a signal.

UCF file

A UCF (user constraints file) contains user-specified logical constraints.