Quick Start Guide for Xilinx Alliance Series 1.5

Quick Start Guide for Xilinx Alliance Series 1.5 — 0401756 01

Printed in U.S.A.

any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx, Inc. will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

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

Quick Start Guide for Xilinx Alliance Series 1.5

Glossary

Preface

About This Manual

This manual provides an overview of the features and additions to the Xilinx Alliance Series release version 1.5.

You must consult *The Programmable Logic Data Book* for device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging. *The Programmable Logic Data Book* is available in hard copy and on the Xilinx web site (http://www.xilinx.com). See http://www.xilinx.com/partinfo/databook.htm for the current version of this book.

For specific design issues or problems, use the Answers Search function on the Web (http://www.xilinx.com/support/searchtd.htm) to access the following.

- Answers Database: current listing of solution records for the Xilinx software tools
- Applications Notes: descriptions of device-specific design techniques and approaches
- Data Sheets: pages from *The Programmable Logic Data Book*
- XCELL Journal: quarterly journals for Xilinx programmable logic users
- Expert Journals: the latest news, design tips, and patch information on the Xilinx design environment

If you cannot access the Web, you can install and access the Answers book with the DynaText® online browser in the same manner as the Xilinx book collection. The Answers book includes information in the Answers Database at the time of this release.

Manual Contents

This manual covers the following topics.

- Chapter 1, "Introduction" introduces the various features of the Xillinx software.
- Chapter 2, "Installing the Software" gives instructions on the installation of the software on workstations, and PCs.
- Chapter 3, "Alliance Series Design Implementation Tools Tutorial" provides a tutorial on the Xilinx design flow.
- Chapter 4, "Using the Software" looks in-depth at the capability and flexibility of the Xilinx software.
- Appendix A, "Cadence Concept and Verilog Interface Notes," covers how to set up the Cadence Concept interface for schematic entry, and Verilog-XL for simulation.
- Appendix B, "Alliance FPGA Express Interface Notes," covers how to install and start using FPGA Express, with the Xilinx Alliance Series Software.
- Appendix C, "Mentor Graphics Interface Notes," covers how to set up the Mentor Graphics interface and associated libraries.
- Appendix D, "Synopsys (XSI) Interface Notes," covers how to set up the Synopsys® interface and associated libraries.
- Appendix E, "Viewlogic Interface Notes," covers how to set up the Viewlogic® interface and project libraries.
- Appendix F, "Using LogiBLOX with CAE Interfaces," covers how to set up the LogiBLOX interface and associated libraries.
- Appendix G, "Instantiated Components," includes a listing of the components most frequently instantiated in synthesis designs.
- Appendix H, "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 I, "Glossary," contains definitions and explanations for terms used in the Quick Start Guide.

Conventions

Typographical

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

- `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*

  Square brackets also enclose footnotes in tables that are printed out as hardcopy in DynaText.

- 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

Xilinx has created several conventions for use within the DynaText 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.
- There are several types of icons.

Iconized figures are identified by the figure icon.

Figure 1-1 Naming Conventions 

Iconized tables are identified by the table icon.

Table 13-14 Carry Modes 

The Copyright icon displays in the upper left corner on the first page of every Xilinx online document.

The DynaText footnote icon displays next to the footnoted text.

Macro

Double-click these icons to display figures, tables, copyright information, or footnotes in a separate window.

- Inline figures display within the text of a document. You can display these figures in a separate window by clicking the figure.

Contents

Chapter 1

Introduction

This chapter contains the following sections.

- "Supported Devices"
- "Supported Netlists"
- "Online Documentation"
- "Xilinx Development System Tools and Features"
- "Third Party Interfaces"

Supported Devices

The Xilinx Development System supports the XC3000A, XC3000L, XC3100A, XC3100L, XC4000E, XC4000L, XC4000EX, XC4000XL, XC4000XV, XC4000XLA, XC5200, XC9500, XC9500XL, Spartan, Spartan XL, and Virtex device families. Refer to *The Programmable Logic Data Book* for more information on these devices.

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

Online Documentation

For a complete list of documentation for this release, refer to the online documentation provided on CD-ROM, or visit the Xilinx Web site at http://www.xilinx.com.

Xilinx Development System Tools and Features

This section lists the tools and the main features of the Xilinx software. The Floorplanner and the Constraints Editor are new for the 1.5 release. For more information on Xilinx tools and features, refer to the appropriate manual in the online documentation.

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.
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.
EPIC	Graphical tool used to display and configure your designs before or after placing and routing.

Table 1-1 Xilinx Software Tools

Feature	Description
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.

Third Party Interfaces

For information on the Xilinx-supplied interface tools, refer to the applicable appendix in this manual, as listed in the following table.

Third-Party Interface	Appendix
Cadence	Appendix A
FPGA Express™	Appendix B
Mentor Graphics	Appendix C

Third-Party Interface	Appendix
Synopsys	Appendix D
Viewlogic	Appendix E

Chapter 2

Installing the Software

This chapter provides a condensed description of the installation process. For detailed information on mounting CD-ROMs and running the various installation programs, see the installation instructions in the Xilinx *Release Document.*

This chapter includes the following sections.

- "Support and Service"
- "Software Licensing"
- "Userware Directory"
- "Requirements for Workstations"
- "Installing Software on Workstations"
- "Requirements for PCs"
- "Installing Software on PCs"

Support and Service

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

Technical Support

If you experience problems with the installation, operation, or verification of your installation, contact the Xilinx Technical Support hotline by phone, e-mail, or fax. When e-mailing or faxing inquiries, provide your complete name, company name, and phone number.

Location	Telephone	Electronic Mail	Facsimile (Fax)
North America	1-408-879-5199 1-800-255-7778	hotline@xilinx.com	1-408-879-4442
United Kingdom	44-1932-820821	ukhelp@xilinx.com	44-1932-828522
France	33-1-3463-0100	frhelp@xilinx.com	33-1-3463-0959
Germany	49-89-93088-130	dlhelp@xilinx.com	49-89-93088-188
Japan	local distributor	jhotline@xilinx.com	local distributor
Korea	local distributor	korea@xilinx.com	local distributor
Hong Kong	local distributor	hongkong@xilinx.com	local distributor
Taiwan	local distributor	taiwan@xilinx.com	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.

Country	Telephone	Facsimile
United States and Canada[1]	1-800-624-4782	408-559-0115
United Kingdom[2]	01932-333550	01932-828521
Belgium[2]	0800 73738	
France[2]	0800 918333	
Germany[2]	0130 816027	
Italy[2]	1677 90403	
Netherlands[2]	0800 0221079	
Other European Locations[2]	(44) 1932-333550	(44) 1932-828521
Japan	81 3 3297 9153	81 3 3297 9189

[1]Mon-Fri, 8:00 am - 5:00 pm Pacific time

[2]Monday–Friday, 9:00 a.m. to 5:30 p.m. United Kingdom time—English speaking only.

Software Licensing

In the current release of the Xilinx software, you do not need a license to run the software. However, you must be a registered user in the Xilinx Customer Service database. If you are a new user, follow the procedure in the next section to register.

When you install the software, you are asked to provide a number located on the back of the Xilinx CD-ROM package. This number instructs the installation program to load either the base or standard software package.

New User Registration

If you are a new Xilinx user, you should fill out your Xilinx registration card and fax or mail it to your Customer Service location.

Userware Directory

The userware directory is installed on your system when you install the Xilinx implementation tools. This directory contains many useful files including the following.

Note: Read the 00_index files in this directory and its sub-directories for a complete list and a description of all files in this area.

- Special utilities for Xilinx software and devices
- *Xilinx Software Conversion Guide* for converting XACT 5.x.x designs to Xilinx Alliance Series 1.5 designs
- Training and presentation material for the current software release
- *Synopsys (XSI) Synthesis and Simulation Design Guide*
- Timing and placement constraints examples (UCF)
- Boundary scan (BSDL) and IBIS models

Requirements for Workstations

This section provides information on software installation workstation requirements.

Supported Architectures and Operating Systems

The current release of the Xilinx software supports the following workstation architectures and operating systems.

- Solaris® 2.5 and 2.6
- HP-UX 10.2
- Ultra Sparc (or equivalent)
- HP715 (or equivalent)
- AIX 4.1.5
- Common Design Environment (CDE)

Memory Requirements

The values given in the following table are for typical designs, and include the loading of the operating system. Additional memory may be required for certain "boundary-case" designs, as well as for concurrent operation of other applications (such as, MS Word or Excel).

Xilinx recommends that you compile XC4000EX designs using an Ultra Sparc, HP715, or equivalent machine type. 64MB of RAM and 64MB of swap space is required to compile XC4000EX designs; however, Xilinx recommends using 128MB of RAM, plus corresponding swap space.

Table 2-1 Memory Requirements for Workstations

Xilinx Device	RAM	Swap Space
XC3000A/L XC3100A/L XC4000E/L XC4028EX through XC4036EX XC4005XL through XC4028XL XC4013XLA through XC4028XLA XC5200 Spartan/XL XC9500/XL	64 MB	64 MB–128 MB
XC4036XL through XC4062XL XC4036XLA through XC4062XLA XCV50 through XCV300	128 MB	128 MB–256 MB
XC4085XL/XLA XC40110XV through XC40150XV XCV400 through XCV600	256 MB	400 MB
XC4020XV through XC40250XV XCV800 through XCV1000	512 MB	800 MB

Installing Software on Workstations

Note: For more detailed information on workstation installation, refer to the "Workstation Installation" chapter of the Xilinx *Release Document.*

To obtain the best performance from the Xilinx software, it is important to install the software correctly on the recommended hardware with the recommended operating memory capacity.

Use the steps in the following checklist as a guide when installing the software. This checklist applies to Solaris 2.5 and 2.6, Ultra Sparc (or equivalent), HP-UX 10.2, HP715 (or equivalent), and AIX 4.1.5. All workstation installations must be done while logged in with "root" authority.

Step	Checklist
1	Be prepared to enter the number on the back of your CD-ROM package when you begin the installation process
2	Install Alliance Series Design Implementation Tools software
3	Install CAE interface and libraries
4	Install online documentation
5	Install CORE Generator (optional)
6	Set environment variables
7	Verify DynaText and variable settings
8	Verify installation

Installing Design Implementation Tools

Follow these steps to install the implementation tools.

1. Mount the CD-ROM(s) labeled "Alliance Series Design Implementation Tools."
2. Run the install program located in the CD-ROM root directory.
3. At the screen prompt, enter the number on the back of your CD-ROM package. This number instructs the installation program to load either the base or standard software.

Installing CAE Interface and Libraries

Follow these steps to install the CAE interface and libraries.

1. Mount the CD-ROM(s) labeled, "Alliance Series CAE Interfaces."
2. Run the install program located in the CD-ROM root directory.

Installing Online Documentation

Note: If you have previously installed the Beta / Pre-release version of the Xilinx software you must re-load the DynaText Browser.

Install the online documentation as follows.

1. Mount the CD-ROM labeled "Alliance Series Documentation."
2. Run the install program located in the CD-ROM root directory.

Installing CORE Generator (optional)

Note: Refer to the "Workstation Installation" chapter of the Xilinx *Release Document* for information on installing the CORE Generator software.

Setting Xilinx Software Variables

You must set various environment variables for the Xilinx software to run correctly.

Set the following variables on HP workstations.

- XILINX
- Path
- LD_LIBRARY_PATH (Solaris only)
- SHLIB_PATH (HP-UX only)

Set these variables as follows.

setenv XILINX *installation_path_of_Xilinx_tools*

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

For Solaris workstations, set the following variables.

setenv LD_LIBRARY_PATH $XILINX/bin/*platform_name***:/usr/openwin/lib**

For HP-UX workstations, set the following.

```
setenv SHLIB_PATH $XILINX/bin/hp:lib:/usr/lib
```

The following is an example for a Solaris workstation.

```
setenv XILINX /usr/xilinx
set path = ($XILINX/bin/sol $path)
setenv LD_LIBRARY_PATH $XILINX/bin/sol:/usr/openwin/lib
```

Setting DynaText Variables

The Xilinx collection of online documentation is located in the $XILINX/doc/usenglish directory. The DynaText Reader Guide is located in the $XILINX/data/dtext directory. To view the documentation, use the DynaText browser included on the Alliance Series Documentation CD-ROM and installed with the online documentation. The browser is installed in the $XILINX/bin/*platform_name* directory. The DynaText environment is defined by the ebtrc file. There is an ebtrc file for each environment supported by the Xilinx software. The environment files are located in the $XILINX/bin/*platform_name* directories.

To use the appropriate setup file, you must set the $EBTRC environment variable. Set this variable as follows.

setenv EBTRC $XILINX/bin/*platform_name***/ebtrc**

The following is an example for a Solaris workstation.

setenv EBTRC $XILINX/bin/sol/ebtrc

In addition to the $EBTRC variable, one of the following workstation variables must be set, depending on your particular platform.

- LIBPATH (for IBM® RS6000)
- SHLIB_PATH (for HP/UX)
- LD_LIBRARY_PATH (for Solaris)

Set the variable as follows.

setenv *platform_variable* **$XILINX/bin/***platform_directory*

To use the DynaText browser enter the following command.

dtext

If the $EBTRC or $XILINX environment variables are not set correctly, DynaText returns the following error.

```
Your .ebtrc does not point to a valid DATA_DIR
```

To fix this error, verify that the environment variable is set correctly and that the path it references can be accessed from the machine running DynaText.

Some warning messages may be displayed on your screen. These warnings indicate missing font information and are due to the platform independence of the DynaText browser. Not all fonts are available on all platforms, as shown in the following example. You can safely ignore these warnings.

```
Warning: Missing charsets in String to FontSet
conversion

Warning: Cannot convert string
"-dt-application-bold-r-*-*-*-140-*-*-p-*-*-*,-sgi-
screen-bold-r-*-*-*-160-*-*-m-*-*-*,-*-
lucidatypewriter-bold-r-*-*-*-120-*-*-*-*-*-*" to
type FontSet
```

Note: For more information on system requirements for running the DynaText browser, and how to make a local copy of the .ebtrc file for customizing, see the "Workstation Installation" chapter of the *Release Document*.

Increasing the Font Size in DynaText (PC™ only)

To increase the font size of the screen text in DynaText on the PC, select the following to display the Preferences dialog box. Increase the value in the Zoom field to enlarge the text on the screen.

File → Preferences

Configuring a Printer for DynaText (Unix Workstation Only)

To define a list of printers, select the following in the DynaText window.

File → Preferences → Printers

You can also configure your browser to print various page sizes and orientations by updating the $XILINX/data/dtext/data/ps/config.dat file. Your System Administrator can update this file.

The config.dat file uses an SGML syntax, with the SPOOLER, FONT, and TYPEFACE elements. The SPOOLER element is used to define both the list of printers and the print orientation. The syntax of the SPOOLER tag is as follows.

<SPOOLER NAME= *PrintSpooler* **WIDTH=** *PointsWide* **HEIGHT=** *PointsHigh* **COMMAND="***SysPrintCmd***">**

These attributes are described in the following table.

Table 2-2 SPOOLER Attributes

Attribute	Value	Description
NAME	*PrintSpooler*	UNIX print-spool name or "default" representing the default printer assigned by the PRINTER environment variable of the process running the browser. The value of this attribute appears in the File → Preferences → Printers menu. This value must be unique.
WIDTH	*PointsWide*	Width in points of the print output (72 points per inch; 28.35 points per centimeter), as in the following examples. 8.5 x 11 paper = 612 points wide* A4 paper = 598 points wide*
HEIGHT	*PointsHigh*	Height in points of the print output, as in the following examples. 8.5 x 11 paper = 792 points high* A4 paper = 845 points high*
COMMAND	*SysPrintCmd*	UNIX printing command for submitting a print job to the specified print spooler.
* The sample values given are for portrait orientation. To print in landscape orientation, switch the values for WIDTH and HEIGHT.		

The following are examples using the SPOOLER element to define the printer.

- Define default printer

```
<SPOOLER name=default width=612 height=792 command="lp">
```

- Define mp1 printer with portrait orientation

```
<SPOOLER name=mp1 width=612 height=792 command="lpr -Pmp1">
```

- Define mp1 printer with landscape orientation

  In this example, the NAME attribute is updated with a more descriptive value. Also, the WIDTH and HEIGHT values are switched.

```
<SPOOLER name=mp1wide width=792 height=612 command="lpr -Pmp1">
```

Verifying Workstation Installation

After the required environment variables are set, you should verify that they are set correctly. Enter the following command to test your setup.

```
par
```

If your setup is correct, PAR runs normally and returns command line information. If a variable is incorrectly set, you may get error messages similar to those in the following table.

Table 2-3 Installation Error Messages and Solutions

Error Message	Solution
`par: Command not found`	Setup has a variable incorrectly set. Check your path and XILINX environment variables.
`ld.so: libbasgi.so.1: not found`	Check the LD_LIBRARY_PATH environment variable if running Solaris, or the SHLIB_PATH environment variable if running HP-UX.
`FATAL ERROR: The XILINX environment variable must be set. Exiting...`	Check the XILINX environment variable.

Setting Up LogiBLOX for Workstations

Refer to the "Using LogiBLOX with CAE Interfaces" appendix of this manual.

Requirements for PCs

This section provides information on software installation PC requirements.

Supported Operating Systems

The current release of the Xilinx software supports the following PC operating systems.

- Windows 95®
- Windows NT® 4.0

Memory Requirements

The values given in the following table are for typical designs, and include the loading of the operating system. Additional memory may be required for certain "boundary-case" designs, as well as for concurrent operation of other applications (such as MS Word or Excel).

Table 2-4 Memory Requirements for PCs

Xilinx Device	RAM	Virtual Memory
XC3000A/L XC3100A/L XC4003E/L through XC4008E/L XC4005XL through XC4008XL XC4000XV XC5202 through XC5210 XC9536 through XC95216 XC9536XL through XC95216XL	32 MB	32 MB–64 MB
XC4010E/L through XC4025E/L XC4028EX through XC4036EX XC4010XL through XC4028XL XC4085XL XC4013XLA through XC4028XLA XC4000XV XC5215 Spartan/XL XC95288/XL	64 MB	64 MB–128 MB
XC4036XL through XC4062XL XC4036XLA through XC4085XLA XC4000XV XC9500/XL	128 MB	128 MB–256 MB
XC40110XV, XC40150XV XCV50, XCV100, XCV150, XCV200, XCV300	128 MB	256 MB
XC40200XV, XC40250XV XCV400, XCV600	256 MB	400 MB
XCV800, XCV1000	512 MB	800 MB

Installing Software on PCs

For more detailed information on PC Installation, refer to the "PC Installation" chapter of the *Release Document*.

Use the steps in the following checklist as a guide when installing the software.

Step	Required Installation, Verification Checklist
1	Be prepared to enter the number on the back of your CD-ROM package when you begin the installation process
2	Install Alliance Series Design Implementation Tools software
3	Install online documentation
4	Install CORE Generator (optional)
5	Set environment variables

Installing Design Implementation Tools

Use the following steps to install the implementation tools.

1. Insert the CD-ROM(s) labeled "Alliance Series Design Implementation Tools" in the CD-ROM drive.
2. Run the setup.exe program in the CD-ROM root directory.
3. At the screen prompt, enter the number on the back of your CD-ROM package. This number instructs the installation program to load either the base or standard software.

Installing Online Documentation

Install the online documentation as follows.

1. Insert the CD-ROM labeled "On-Line Documentation" in the CD-ROM drive.
2. Run the setup.exe program in the CD-ROM root directory.

Installing CORE Generator (optional)

Note: Refer to the "PC Installation" chapter of the *Release Document* for information on installing the CORE Generator software.

Setting Environment Variables

You must set the XILINX and PATH variables for the Xilinx software to run correctly. Set these variables as follows.

```
set XILINX=c:\xilinx
set PATH=c:\xilinx\bin\nt;%PATH%
```

These variables are set to the C:\XILINX directory, which is the default path for the Xilinx Implementation Tools. If you change the default path, you must change the environment variables to match the new path.

Setting Variables with Windows NT

To set the XILINX and PATH variables on a PC with Windows NT 4.0, follow these steps.

1. Select **Start** → **Settings** → **Control Panel**.
2. Double click on the System icon and select the Environment tab. Verify that the settings shown previously are listed in either the System Variables section or in the User Variables section. They may not appear exactly as previously shown; the variable may be under the Variable header and the path may be under the Value header. The word "set" does not appear.

Setting Variables with Windows 95

To set the XILINX and PATH variables on a PC with Windows 95, follow these steps.

1. Run SYSEDIT to open the AUTOEXEC.BAT file.
2. Verify the environment settings are as previously shown.

Chapter 3

Alliance Series Design Implementation Tools Tutorial

Note: An updated tutorial will be available after June 30, 1998 from the Xilinx web site and on the AppLINX CD. The web site location is http://www.xilinx.com/support/techsup/tutorials. Contact your local sales representative for a copy of the AppLINX CD.

This chapter contains the following sections.

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

Installing the Tutorial

This tutorial demonstrates the Alliance Series Design Implementation Tools flow. The tutorial design is a simple 8-bit counter with asynchronous clear and clock enable. The design is compiled with Synopsys FPGA Express and is described by a Xilinx Netlist File

(XNF). The tutorial passes an input netlist from FPGA Express to the Alliance Series Design Implementation Tools, and incorporates timing and placement constraints through a User Constraints File (UCF).

First, create an empty working directory. In this tutorial, the newly created directory is named Count. Next, copy the following files from the /userware/tutorial/qstart/ directory located on the Alliance Series Design Implementation Tools CD-ROM to your newly created working directory.

Note: The source code (count8.vhd) is also available for reference.

File Name	Description
count8.xnf	Xilinx Netlist File
count8.ucf	User Constraints File

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 as well as the implementation revisions within each version. The Design Manager also provides access to the entire suite of Xilinx implementation tools you need to complete a design.

1. To begin this tutorial, change directories to the area containing your copy of the design files.
2. On a workstation, enter the following at the command line prompt to start the Design Manager.

   ```
   xilinx &
   ```

   You can also start the Design Manager with the following command.

   ```
   dsgnmgr &
   ```

   On a PC, start the Design Manager by entering.

   Start → Programs → Xilinx → Design Manager

3. When running the Design Manager for the first time, there are no projects available. To create a new implementation project for this tutorial design, enter the following.

   `Select File → New Project`

   The New Project dialog box appears containing fields to specify the input design, working directory, and a design comment. The input design is the top level netlist file that contains the design's definition. The working directory is the area that will be used by the tools to store the implementation data created as you compile your design. Use the comment field to enter a brief notation about the design being processed.

4. Click on the Browse button to the right of the Input Design Field to specify the input design.

   The Browse dialog box is displayed with the default file type set to EDIF Files. The design for this tutorial was created by Synopsys FPGA Express, which outputs an XNF file.

5. Click on the List Files of Type pull-down list box and select XNF Files (*.xnf, *.xtf, *.sxnf) to change the file filter to the desired file type.

6. Select the count8.xnf file and click on OK to accept the input netlist.

Note: You may need to change directories to the area containing the copied design files.

   The Browse dialog box is closed and the New Project dialog box is updated to include the specified input netlist. By default, the Work Directory is set to the directory containing the input design. If you prefer, you can set this to another directory. Because the files were previously copied into the Count directory, this same directory is used to hold the implementation project and resulting output files.

7. Place the cursor in the Comment field and enter the following.

   `-tutorial`

8. Click OK to close the New Project dialog box and to update the Design Manager with the specified project, as shown in the following figure.

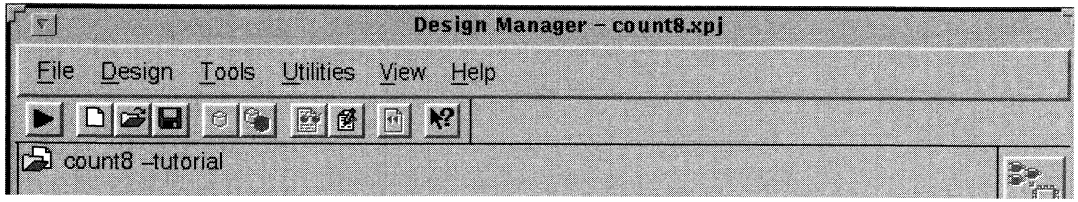


Figure 3-1 Count8.xpj Project in Design Manager

Note: The toolbox buttons on the right side of the Design Manager are inactive. To use these tools, you must first create a design version and an implementation revision.

Step 2: Specifying Options

Each time a change is made to the input design, a new design version must be created in the Design Manager. You can then use the tools to create as many implementation revisions as you like for that design version. For example, when attempting to try different implementation strategies on a given netlist, you are creating several revisions for a singe version. By default, the Design Manager only keeps track of the Xilinx-created files. The input design is not stored with the implementation data in the Xilinx implementation project area.

Note: This tutorial describes the basic flow. For detailed information on flows and implementation methodologies using the Alliance Series Design Implementation Tools, see the online version of the *Development System Reference Guide.*

1. Select **`Design`** → **`Implement`**

   The Design Manager prompts for the correct device by displaying the Implement window. Notice that the Part field already contains the value 'XC4002XL-09-PC84'. This information was read by the Design Manager from the input netlist (count8.xnf). The Implement dialog box is also displayed with the default version, 'ver1', and revision, 'rev1', to be created.

2. Select Options to open the Options dialog box.

   The Options dialog box allows you to specify a user constraints file and any optional processing targets. You can also access three types of templates (implementation, simulation, and configuration).

Templates provide a convenient way to have several groups of option settings that you can select from when you implement a design. The available options depend on the target device family. For example, you can have a template for quick place and route and another to implement a certain configuration option.

3. Verify that the entire path to the count8.ucf file is specified in the User Constraints field.

   When initially creating a project, if you include a user-generated UCF file in the same directory as your input design file, then the UCF file automatically appears in this field.

   You can open the count8.ucf file with a text editor to view the constraints specified for this particular design. For a detailed explanation of constraints as well as examples with proper syntax, refer to the Xilinx *Libraries Guide.*

4. Select Edit Template for the Implementation Program Option.

   The XC4000 Implementation Options dialog box is displayed. The implementation templates control how the software maps, places, routes, and optimizes a design.

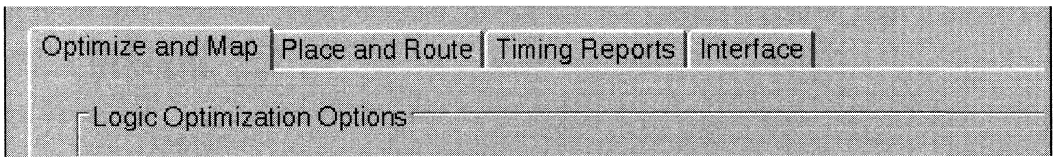


Figure 3-2 XC4000 Implementations Options Dialog Box

5. Select the Timing Reports tab.
6. Select Produce Logic Level Timing Report. The option to Produce Post Layout Timing Report should be selected by default.

   The timing reports, generated after MAP and PAR, are useful in evaluating design performance. They are analyzed in detail later in this tutorial.

7. Click OK once to save the Implementation options, then again to exit the Options dialog box.

Step 3: Implementing the Design

Based on the specified constraints and the selected report files, you can now implement the design. The first stage of implementation is Translate, which runs NGDBuild. NGDBuild performs the following functions.

- Converts input design netlists and writes the results to a single merged NGD file.
- Adds the User Constraints File (UCF) to the merged netlist. The merged netlist describes the logic in the design and any location and timing constraints.

1. Click Run in the Implement dialog box to begin processing.

   This starts the Flow Engine to implement the design. Stages or processes in the design are graphically represented in the upper half of the Flow Engine window. The status of each stage is also depicted here.

2. In this tutorial, you want to stop after translation. To do this, you must set a break point to stop the Flow Engine at this point. To stop the program, click on the stop sign toolbar icon while Translate is running. Refer to the following figure.

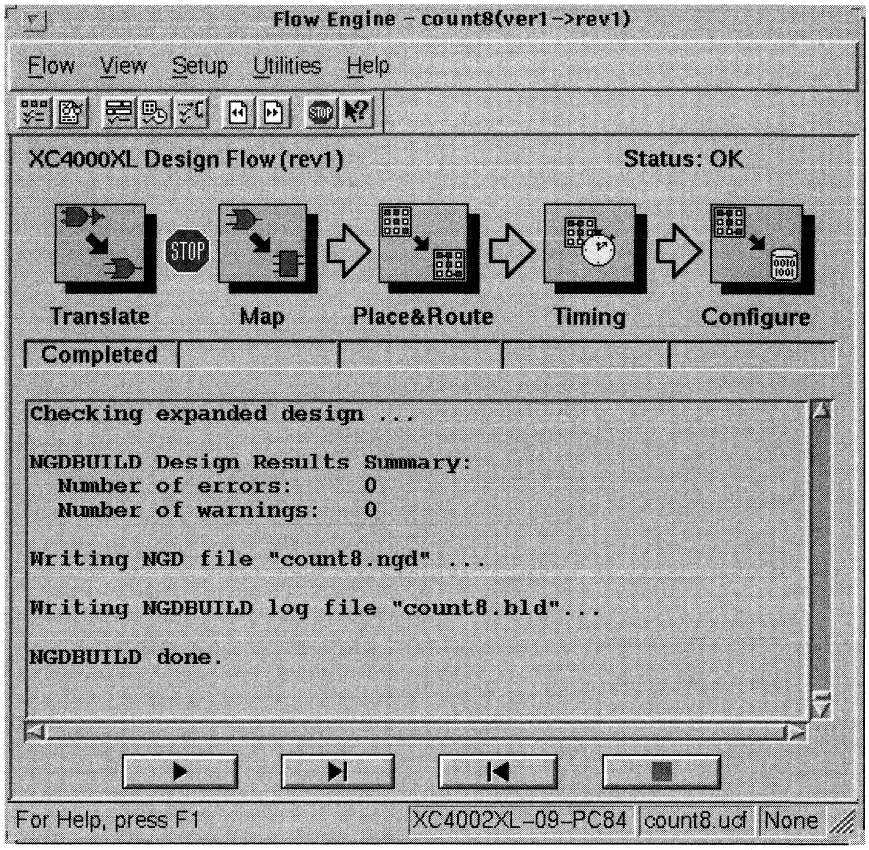


Figure 3-3 Translating Design

The Stop After dialog box is displayed with the default setting of Configure. The list box only contains possible break points with respect to the current state of the design. Because the design is not processed yet, all possible break points are available.

3. Select Translate in the list box and click on OK to accept the break point. On your screen, the stop sign is now added to the flow between the Translate and Map phases. This stop allows you to run the flow until the break point.

Note: The status bar at the bottom of the Flow Engine is updated with the specified user constraints file (count8.ucf).

Setting a break point after the Translate phase is useful when you want to perform a functional simulation of the design and copy out the resulting *design*.ngd file to your working directory. Once you have the *design*.ngd file copied, 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*.

4. After Translate is done, an Implement Status dialog box appears. Select OK.

   The Design Manager shows 'rev1' under the initial version of the count8 project. The status of the revision is noted as (Translated, OK). *Translated* 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. So far, the design has not produced any errors.

   At the bottom of the Design Manager is the status bar. The status bar contains information such as the current project, target device, and currently selected version → revision pair. The left-hand portion of the status bar is used to provide information on what is currently selected by the cursor.

Figure 3-4 Design Manager Status Bar

The toolbox, located on the right side of the Design Manager, becomes active with your first implementation revision. The icons contained in the toolbox are only active when a revision is selected. Icons in the toolbox (as shown in the following figure) represent the Flow Engine, Timing Analyzer, Floorplanner, PROM File Formatter, Hardware Debugger, and EPIC Design Editor.

Note: The toolbar has drag and drop capability.

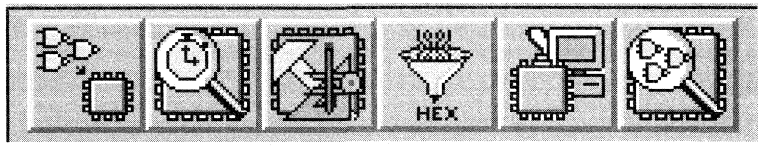


Figure 3-5 Design Manager Toolbox

Step 4: Mapping the Design

The Design Manager manages the files created during the implementation process while the Flow Engine controls the implementation process itself. All the programs that run on a design are actually run by the Flow Engine based on the settings you supply in the various dialog boxes and templates.

1. Click the Flow Engine icon in the toolbox on the right side of the Design Manager.

   The Flow Engine is invoked using the flow settings specified earlier. The first stage of translating the input netlist and merging it into a single design file is complete. The design is now ready to be mapped into CLBs and IOBs. After mapping, the design is placed and routed. Finally, the configure stage creates a configuration bitstream that can be downloaded to the target system or formatted into a PROM programming file.

   The Flow Engine gives you complete control over how a design is processed. Typically you set all desired implementation options and run through the entire flow. However, to demonstrate the flexibility of the Flow Engine, and provide you with a working knowledge of the tool, this tutorial proceeds through each stage of the flow. For the Xilinx design flow, refer to the "Xilinx Design Flow" figure of the "Using the Software" chapter.

Note: There are several ways to begin the implementation process. You can use the Flow → Run and Flow → Step commands, or you can use the equivalent control buttons, as shown in the following figure.

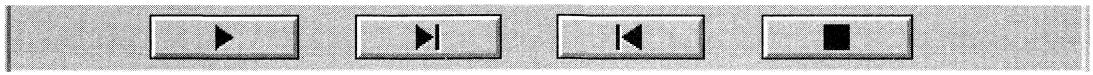


Figure 3-6 Control Panel

2. Click the Step control button to start the Map process.

   Even though a break point is not specified, the Flow Engine stops after the Map stage because only a single step is being run.

   Map performs the following functions.

   - Allocates CLB and IOB resources for all the 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

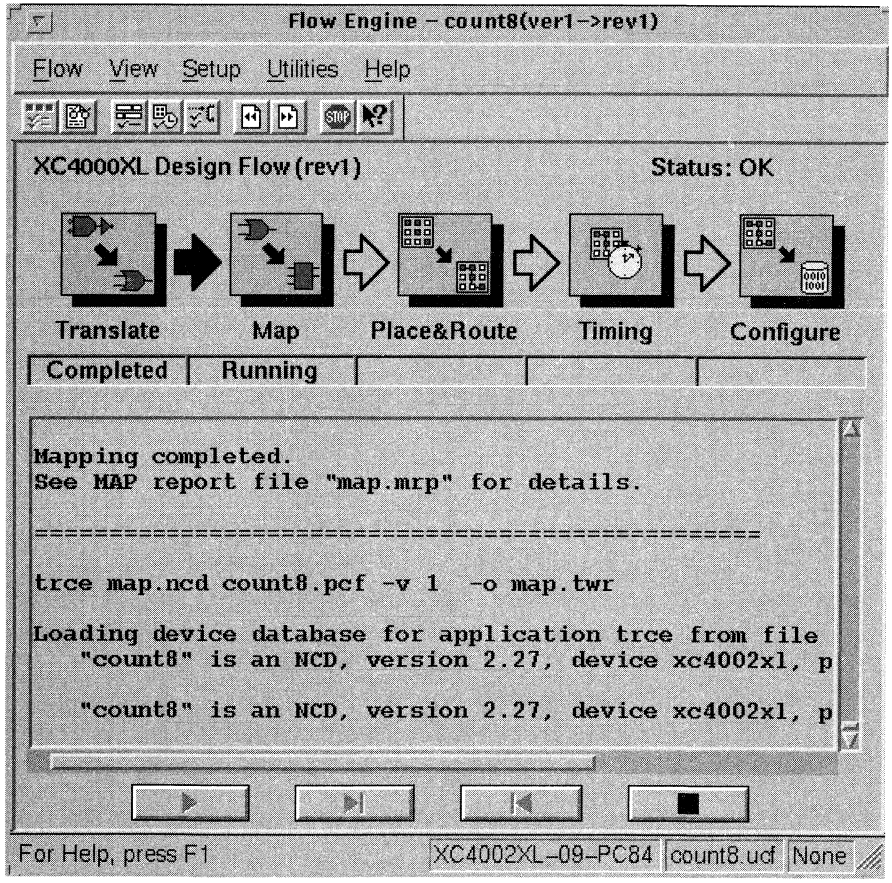


Figure 3-7 Mapping Design

3. While Map is running, you can review the currently available reports using the Report Browser by selecting the following.

   `Utilities → Report Browser`

   The Report Browser is displayed with the Translation report. The Map and Logic Level Timing Report files are created after Map is finished. New reports are denoted with a gold star in the upper left corner of the file icon.

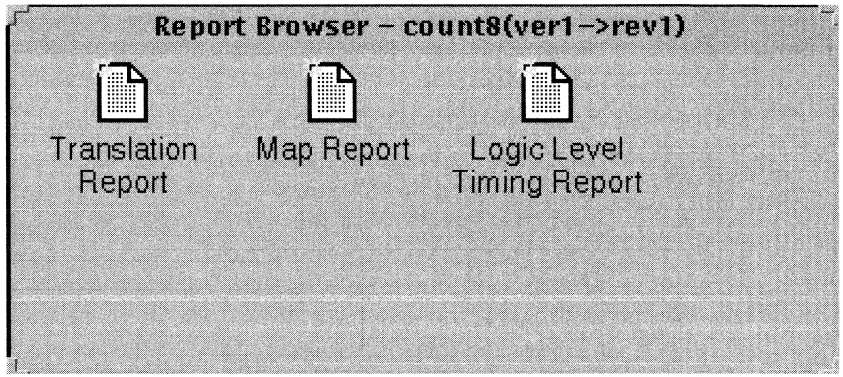


Figure 3-8 Report Browser after Running Map

4. Double-click on a report to review its output.

   - Translation Report — contains warning and error messages from the translation process. The translation process converts the input design netlist to the Xilinx NGD netlist as well as performs timing specification and logical design rule checks.
   - Map Report — contains 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 place and route.

   At this point in the Design Manger, the current version → revision should have the status (Mapped, OK). The design has been mapped to the target architecture. The next step is evaluating

paths in the design to ensure that they do not contain excessive block delays.

Step 5: 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 available. The timing report describes the logical block delays and routing delays. The net delays that are provided are based on an optimal distance between blocks (also referred to as *unplaced floors*).

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 particular path will make up about 50% of the total path delay once the design is routed. This means that a path containing 10ns of block delay should meet a 20ns timing constraint after it has been placed and routed. If you have a high density design or use an architecture with fewer routing resources, your net delays may take more than 50% of the total path delay.

The following section from the Logic Level Timing Report represents a single path delay covered by the timing constraint specified in the user constraints file. This excerpt contains the following worst case path.

```
================================================================================
Timing constraint: TS_CLOCK = PERIOD TIMEGRP "CLOCK"  20 nS HIGH 50.000% ;
 44 items analyzed, 0 timing errors detected.
 Minimum period is   6.352ns.
--------------------------------------------------------------------------------
Slack:     13.648ns path N91 to N86 relative to
           20.000ns delay constraint

Path N91 to N86 contains 5 levels of logic:
Path starting from Comp: CLB.K (from CLOCK_BUFGed)
To                      Delay type         Delay(ns)  Physical Resource
```

```
                                                        Logical Resource(s)
-------------------------------------------------  --------
CLB.YQ                  Tcko                 1.470R  N91
                                                        QOUT_reg<0>
CLB.G4                  net (fanout=2)     e 0.342R  N91
CLB.COUT                Topcy                1.600R  N91
                                                        C17_C0_C1
CLB.CIN                 net (fanout=1)     e 0.230R  C17_N2
CLB.COUT                Tbyp                 0.140R  N90
                                                        C17_C1_C2
CLB.CIN                 net (fanout=1)     e 0.230R  C17_N7
CLB.COUT                Tbyp                 0.140R  N88
                                                        C17_C2_C2
CLB.CIN                 net (fanout=1)     e 0.230R  C17_N12
CLB.K                   Tsumc+Tick           1.970R  N86
                                                        C17_C3_C2
                                                        c17_c3_c1_c0
                                                        QOUT_reg<6>
--------------------------------------------------------------------------------
Total (5.320ns logic, 1.032ns route)          6.352ns (to CLOCK_BUFGed)
      (83.8% logic, 16.2%% route)
--------------------------------------------------------------------------------
```

Once a design is mapped, you can roughly determine design performance by applying the 50/50 rule. For example, for the previous report excerpt, if you apply the 50% block (logic delay), 50% routing delay rule, the worst case path should be about 10ns after routing the design, given that the block levels currently contribute about 5ns (5.320ns). The timing constraint for this portion of the design is 20ns, leaving a slack time of 13.648ns. Therefore, this portion of the design is well within the timing specification.

The unplaced floors listed are actually estimates (indicated by the letter "e" next to the net delay) based on optimal placement of blocks. The total unplaced floors estimate for the design equals 1.032ns (3 x .23 + .342).

To obtain an even more accurate assessment of timing requirements, you can place a design without routing it. The resulting placed floors, which are based on optimistic routing estimates, provide an even more realistic account of net delays than the unplaced floors estimates from the mapped design. The placed estimates calculate an absolute minimum routing for each placement. For detailed information on how to do this, refer to the *Development System Reference Guide.*

If you do not generate a Logical Level Timing Report, PAR (place and route program) 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 totaling 7 ns and unplaced floor net delays totalling 3 ns, PAR stops and generates an error message. In this case, PAR determines that the total delay (10 ns) is greater than the constraint placed on the design (8 ns), and appropriately fails. The Logic Level Timing Report provides an opportunity to determine any timing violation that may occur prior to running PAR.

Step 6: Placing and Routing the Design

After the mapped design is evaluated to verify that block delays are reasonable given the specified timing goals, you can use the Flow Engine to place and route the design.

The Flow Engine can run the place and route algorithms as follows.

- Timing Driven — run PAR with timing constraints specified from within the input netlist and / or a constraints file
- Non-Timing Driven — instruct the place and route tools to ignore all timing constraints

In this tutorial, timing-driven placement and timing-driven routing are automatically executed by PAR according to the timing constraints specified earlier in the implementation process.

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

1. Close the Report Browser if necessary.
2. Select the following to run only the Place & Route phase.

   `Flow → Step`

   Refer to the following figure.

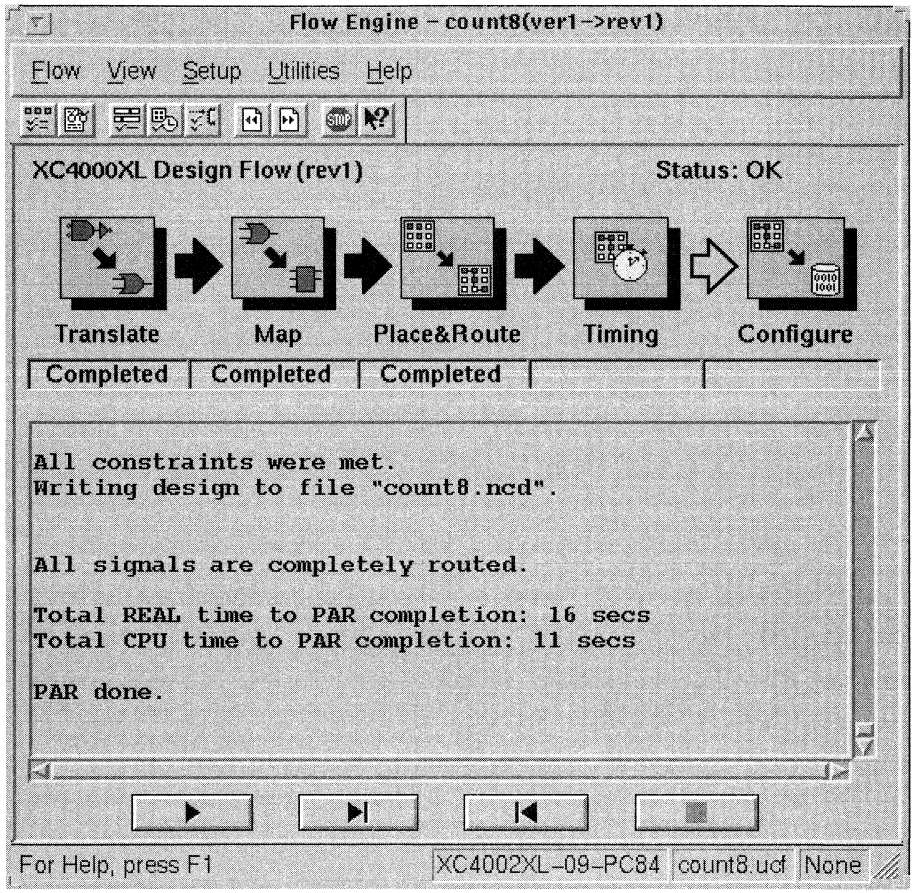


Figure 3-9 Placing and Routing Design

3. Review the reports to make sure the place and route process finished as expected. The Status:OK message in the upper right corner of the Flow Engine indicates that no errors were encountered. Select the following to start the Report Browser, as shown in the following figure.

   `Utilities → Report Browser`

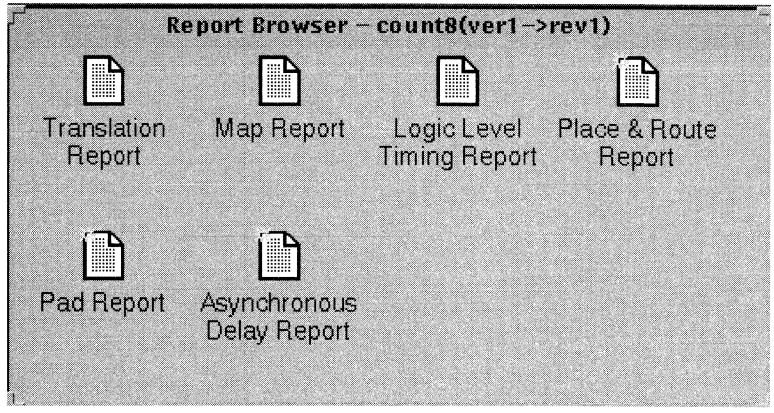


Figure 3-10 Reports Available After Place & Route

The three new reports are the Place and Route Report, the Pad Report, and the Asynchronous Delay Report.

- 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 — enumerates all the nets in the design and the delays of all the loads on the net.

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

Step 7: Evaluating Post-Layout Timing

After the design is placed and routed, you can generate the Post-Layout Timing Report to verify that the design meets the desired

timing goals. This report evaluates both the logical block delays as well as the routing delays. The net delays are now reported as actual routing delays after the place and route process.

Because the option to produce a Post-Layout Timing Report was previously selected, you can proceed with the Timing stage of the design process.

1. From within the Flow Engine, select the following to run only the Timing phase.

   Flow → Step

2. Once completed, select the following.

   Utilities → Report Browser

   Open the newly created Post-Layout Timing Report. At this point, the minimum period for the design is 7.121ns, still well within the timing constraint of 20ns.

3. To obtain a more detailed analysis of post-layout timing, select the following from the Design Manager to start the Timing Analyzer tool.

   Tools → Timing Analyzer

   The Timing Analyzer performs static timing analysis on designs that are mapped and partially or completely placed, routed or both. It organizes and displays data so you can analyze critical paths in your circuit, circuit cycle times, delays on specified paths, and paths with the greatest delay.

   In this tutorial, the Timing Analyzer automatically loads the placed and routed netlist as well as a physical constraints file (PCF). The PCF file contains the same constraints specified in the UCF file, however now expressed in terms of physical elements.

4. Select the following.

   Analyze → Timing Constraints → Report Paths in Timing Constraints

   The Timing Analysis In Progress dialog box is displayed while a verbose report is generated. Once complete, the report automatically appears in the Timing Analyzer.

   After the Map process, logic delay contributed to 5.32 ns of the minimum period attained. The analysis report indicates that this

value has not changed. However, the total unplaced floors estimate of 1.032 ns has changed. The routing delay after PAR equals 1.801 ns (2 x .23 + .236 + 1.105). This is now a true report of net delays after the place and route stage.

The post-layout result does not necessarily follow the 50/50 rule described previously because the worst case path is made up of mainly component delays. After mapping the design, block delays constituted about 80% of the period. Following place and route, the worst case path is still almost 75% logic delay. Since there is only 1.801ns of total routing delay spread out across four nets, expecting this to be reduced any further is not reasonable, especially considering that routing delay only makes up 25% of the total path delay.

Because the timing requirements are not overly aggressive for this design, no timing errors are generated. If a design failed to meet a timespec, one solution is to reduce logic levels in order to reduce block delays.

5. Select the following to close the Timing Analyzer.

   File → Exit

   At this point, you can either save the generated report or discard the results. For information on using more advanced features in the Timing Analyzer, refer to the *Timing Analyzer Reference/User Guide.*

Step 8: Creating Timing Simulation Data

With the design placed and routed and the timing statically verified, the next step is to create timing simulation data for the design. The Modelsim VHDL is selected for the simulator. VHDL is targeted as the back-annotated format for the simulation data.

1. To create VHDL format simulation data, perform the following steps in the Flow Engine.

   a) Select Setup → Options to open the Options dialog box.

   b) Select Modelsim VHDL from the Simulation Option Template pull-down menu.

   c) Click on the Produce Timing Simulation Data option.

   d) Click OK to close the Options dialog box.

2. Select Flow → Step Back from the Flow Engine menu to backup to the Place & Route Completed stage.
3. Select Flow → Step to run Timing.

Note: The Timing phase of the Flow Engine produces timing simulation data. This stage is necessary because the option to produce timing simulation data was not selected in the initial pass.

During Timing, the Flow Engine runs the NGDAnno program to create a back-annotated NGD file. The NGD file is then used as the input to one of the NGD2XXX programs to produce the preferred simulation file format. Because Modelsim VHDL was specified, the Flow Engine runs NGD2VHDL and creates, by default, the files tim_sim.vhd and tim_sim.sdf. The first file is a structural VHDL file and the second is a Standard Delay Format file. To make it easy to find these files to use in a third party simulation environment, they are automatically copied to the working directory.

Step 9: Using the Flow Engine to Create Configuration Data

The next step is creating configuration data. First, you must create a bitstream for the target device by running the Configure phase in the Flow Engine.

1. If the Flow Engine is not running, invoke it on the Timed revision created in the previous step.
2. Select Setup → Options to open the Options dialog box.
3. Select Edit Template for the Configuration Program Option.

   The XC4000 Configuration Options dialog box is displayed. The configuration templates set options that define the initial configuration parameters, the start-up sequence, readback capabilities, and other advanced features. In this tutorial, you will set the input and output threshold levels to CMOS.
4. In the Configuration tab, click on the CMOS radio boxes in the Threshold Levels area for both inputs and outputs.
5. Click OK to close the XC4000 Configuration Options dialog box.
6. Click OK to close the Options dialog box.

7. Select Flow → Run to run the Configure phase.

   The Flow Engine runs BitGen to create the configuration data. BitGen creates the count8.bit and count8.ll files. The count8.bit file is the actual configuration data. The count8.ll file is the logical allocation file that is used during hardware debugging to determine the location of the probable points in the design. The *design*.bit and *design*.ll files are automatically copied to the working directory. Verify that they are in this directory.

   The *design*.11 file is used to perform device readback with the Hardware Debugger. For more information on device readback, see the *Hardware Debugger User Guide*.

   The Flow Engine saves the configuration options in the BitGen Report. Review the report using the Report Browser and verify that the CMOS thresholds option was specified when creating the configuration data. The following figure shows the Configure stage after it is finished.

8. Select Flow → Close to close the Flow Engine and the Report Browser.

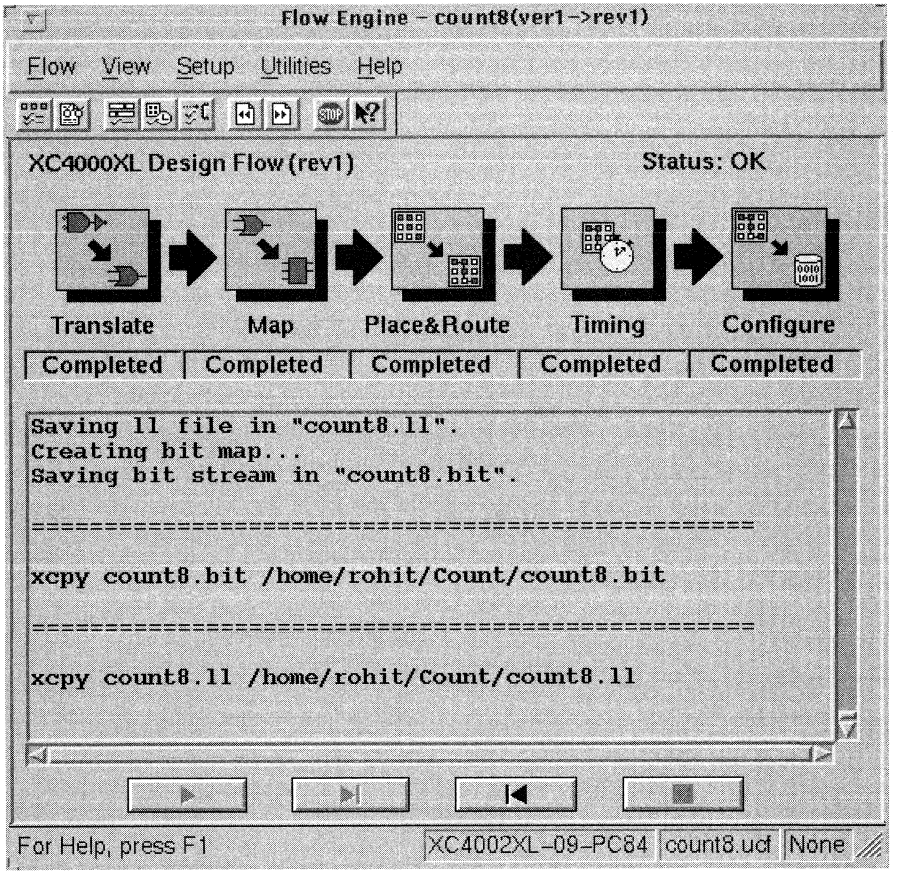


Figure 3-11 Configuring Design

Step 10: Using the PROM File Formatter

If you are only 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 takes in any number of bitstreams and creates one or more PROM files containing one or more daisy chain configurations.

1. To invoke the PROM File Formatter, click on the PROM File Formatter icon in the toolbox from within Design Manager.

   The PFF is invoked 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 from scratch, or immediately save the current PROM file configuration.

   The status bar at the bottom of the PFF indicates 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. Because the target device for this tutorial is an XC4002XL, 60,948 bits of data are required to hold the configuration bitstream. The PFF determined that an XC1765D is the correct PROM because it can hold up to 65,536 configuration bits (or 93% full).

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

2. Select File → PROM Properties to open the PROM Properties dialog box, as shown in the following figure.

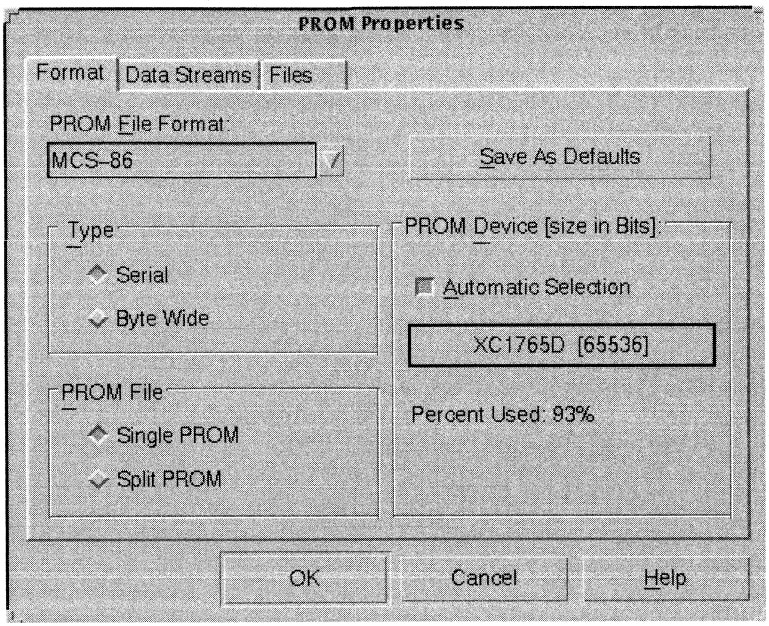


Figure 3-12 PROM Properties Dialog Box with Single PROM

Use the PROM Properties dialog box to select the PROM format, the PROM type used, and the 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.

3. Select File Æ Save to save the PROM file.
4. Specify the working directory as the area where the PROM Description File will be saved.

   The PROM File Formatter saves both the PROM file (count8.mcs) and a PROM Description File (count8.pdr). The PDR file can be re-opened if any changes are required. Verify that the files exist in your directory.
5. Select File → Exit to close the PROM File Formatter.

   This completes the tutorial.

Chapter 4

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"

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.

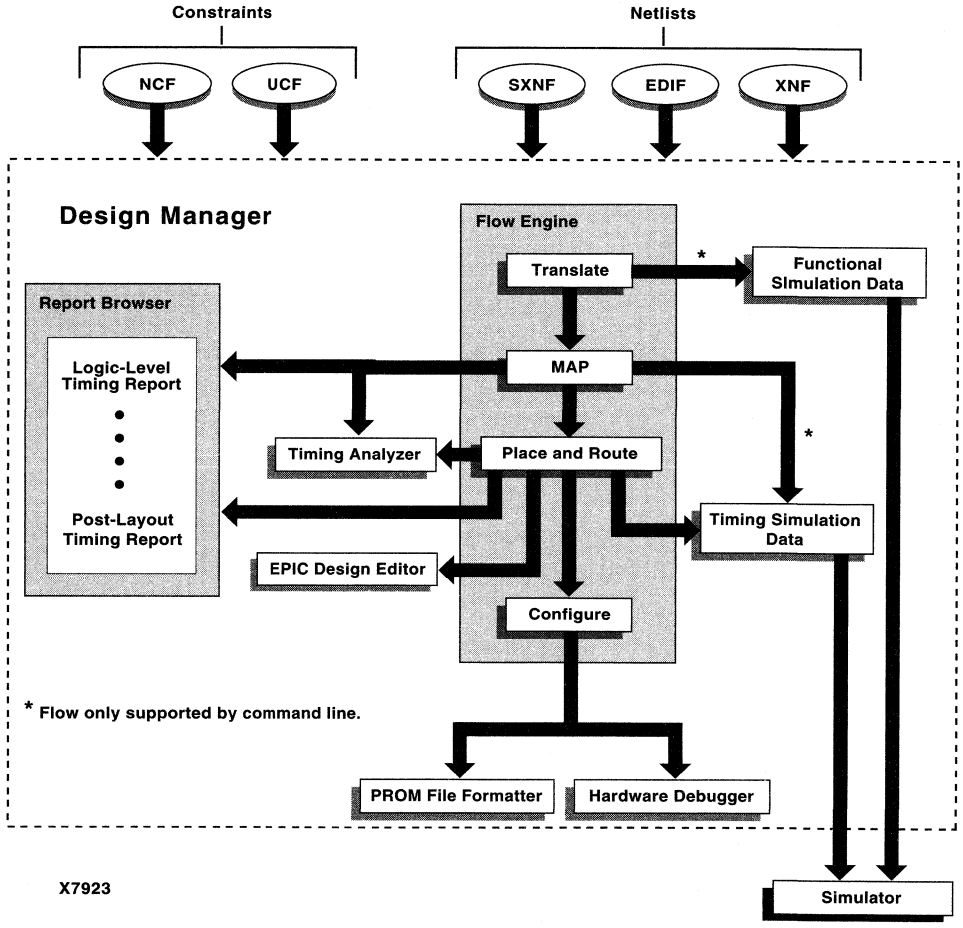


Figure 4-1 Xilinx Design Flow

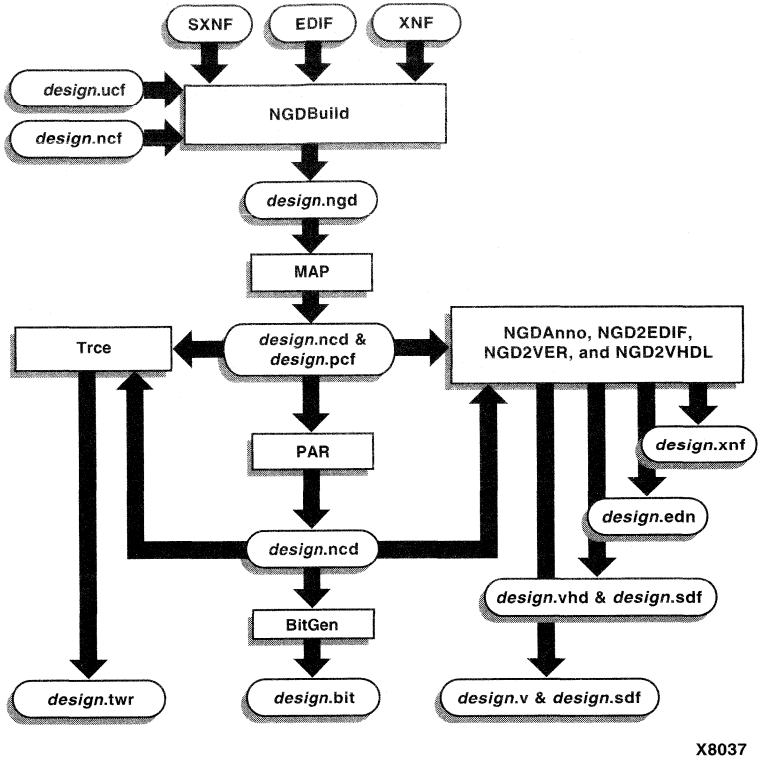


Figure 4-2 Detailed Design Flow

Using the Design Manager

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

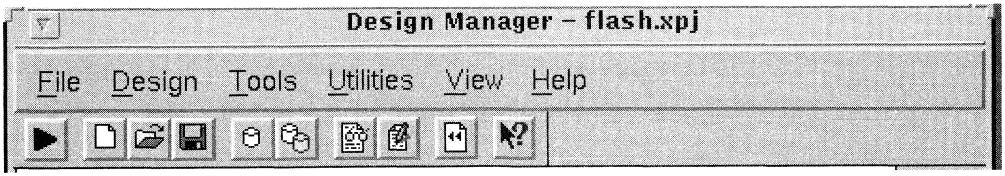


Figure 4-3 Design Manager Menu

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 following appropriate appendix.

- "Cadence Concept and Verilog Interface Notes" appendix
- "Alliance FPGA Express Interface Notes" appendix
- "Mentor Graphics Interface Notes" appendix
- "Synopsys (XSI) Interface Notes" appendix
- "Viewlogic Interface Notes" appendix
- "Instantiated Components" appendix

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 on 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.

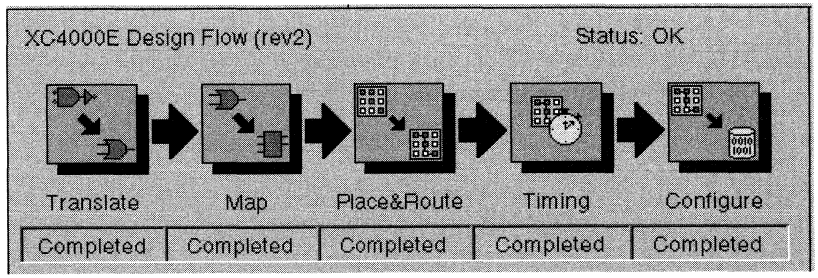


Figure 4-4 Flow Engine Design Steps

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.

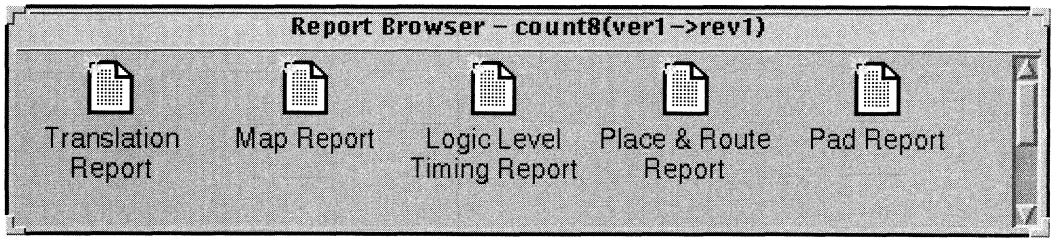


Figure 4-5 Report Browser

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 architecture-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.

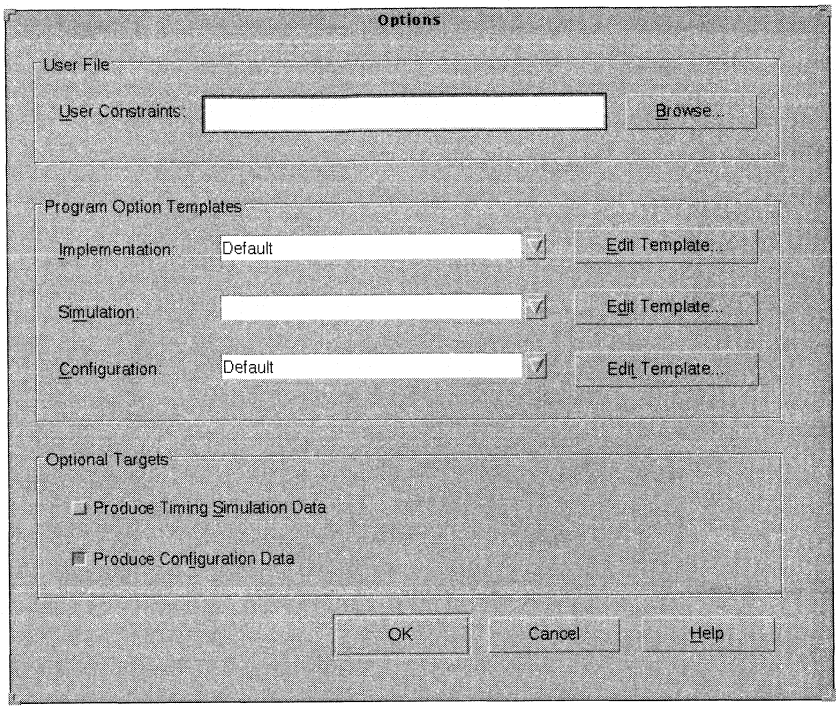


Figure 4-6 Options Dialog Box

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 Reference/User 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
- 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 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).

Starting the Constraints Editor

To start the Constraints Editor, select the following in the Design Manager.

`Utilities → Constraints Editor`

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.

- 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
- To modify the reports to detail path delays or paths failing timing constraints.
  - Edit the Implementation template
  - Select the Timing Reports tab
  - Select a report format
- 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.

- Choose sources. From the Timing Analyzer menu select.

  Path Filters → Path Analysis Filters → Select Sources

- Choose destinations. From the Timing Analyzer menu select

  Path Filters → Path Analysis Filters → Select Destinations

- To create a report, select.

  Analyze → All Paths

To switch speed grades.

- **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 (Generic, ViewLogic, Mentor®, or Foundation). 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 the 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 the simulation vendor (Synopsys, Viewlogic, Mentor Graphics, and Cadence) and the Xilinx interface software. 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 Hardware Debugger can download a bit file or a PROM file: MCS, EXO, or TEK.

For more information on downloading the Hardware Debugger or the XChecker cable, see the *Hardware Debugger Reference/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 Reference/User 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.

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

Appendix A

Cadence Concept and Verilog Interface Notes

This appendix provides information on setting up the Cadence Concept interface for schematic entry, and Verilog-XL for simulation. Included are recommendations on methods for locking pins and entering timing constraints. This appendix contains the following sections.

- "Documentation"
- "Setting Up the Cadence Interface"
- "Cadence/Verilog Design Flow"
- "Setting Up for Concept"
- "Using HDL Direct"
- "Iterated Instances Versus Size Support"
- "Starting Concept"
- "Functional Simulation"
- "Translating a Design to EDIF"
- "Timing Simulation"
- "Support for Board Level Simulation"
- "Pin Locking"
- "Timing Constraints"

Documentation

The following documentation is available for the Cadence interface.

- The Xilinx *Cadence Interface/Tutorial Guide* is available on the Alliance Series Documentation CD-ROM supplied with your software.
- The Cadence software documentation (for Cadence applications such as Concept and Verilog-XL) is available from Cadence in an online format. You can view it by entering the following on the UNIX command line.

  openbook

- The Xilinx *Release Documentation* describes current issues regarding the use of the Cadence interface.

Setting Up the Cadence Interface

In addition to the environment variables described in the "Installing the Software" chapter, the following environment variables must be modified or added to run the Cadence interface tools.

- CDS_INST_DIR (add for Concept)
- VERILOGEXE (add for Verilog-XL)
- XAPPLRESDIR (add for Verilog-XL)
- XNLSPATH (modify for Verilog-XL)
- XKEYSYMDB (modify for Verilog-XL)
- path (modify)
- LD_LIBRARY_PATH (modify for Solaris)
- SHLIB_PATH (modify for HP/UX)
- LIBPATH (modify for IBM RS6000)

Set these variables as follows.

setenv CDS_INST_DIR *installation_path_to_cadence*

```
setenv VERILOGEXE $CDS_INST_DIR/tools/verilog/bin/
verilog
```

```
setenv XAPPLRESDIR $CDS_INST_DIR/tools/verilog/etc
```

setenv XNLSPATH $CDS_INST_DIR/tools/verilog/etc/nls:$XNLSPATH

setenv XKEYSYMDB $CDS_INST_DIR/tools/verilog/etc/XKeysymDB:$XKEYSYMDB

set path = ($CDS_INST_DIR/tools/bin $CDS_INST_DIR/tools/pic/picdesigner/bin $CDS_INST_DIR/tools/editor/lib $CDS_INST_DIR/tools/dfII/bin $path)

For Solaris only.

setenv LD_LIBRARY_PATH $CDS_INST_DIR/tools/lib:$CDS_INST_DIR/tools/verilog/lib: *path_to_x11_libs*:**/usr/lib:$OPENWINHOME/lib:$LD_LIBRARY_PATH**

For HP/UX only.

setenv SHLIB_PATH $CDS_INST_DIR/tools/lib:$CDS_INST_DIR/tools/verilog/lib:/usr/lib:/lib: *path_to_x11_libs*: **$SHLIB_PATH**

For IBM RS6000 only.

setenv LIBPATH $CDS_INST_DIR/tools/lib:$CDS_INST_DIR/tools/verilog/lib:/usr/lib:/lib:$LIBPATH

It is common to create a soft link called "tools" under $CDS_INST_DIR, and to link it to the directory $CDS_INST_DIR/tools.<*platform*>, where *platform* is "hppa" (for HP7), "sun4v" (for Solaris), or "ibmrs" (for IBM RS6000). If your Cadence tool directory is not set up in this way, then substitute "tools.*platform*" where you see "tools" previously, as shown in the following example.

```
setenv CDS_INST_DIR /products/cds.ver97a

setenv VERILOGEXE $CDS_INST_DIR/tools/verilog/bin/verilog

setenv XAPPLRESDIR $CDS_INST_DIR/tools/verilog/etc

setenv XNLSPATH $CDS_INST_DIR/tools/verilog/etc/nls:$XNLSPATH

setenv XKEYSYMDB $CDS_INST_DIR/tools/verilog/etc/XKeysymDB:$XKEYSYMDB

setenv LD_LIBRARY_PATH $CDS_INST_DIR/tools/lib: $OPENWINHOME/lib : /usr/
lib :/tools/x11r5/sun5/lib: $LD_LIBRARY_PATH

set path = ($CDS_INST_DIR/tools/bin \
$CDS_INST_DIR/tools/pic/picdesigner/bin \
$CDS_INST_DIR/tools/editor/lib \
```

```
$CDS_INST_DIR/tools/dfII/bin \
$path)
```

Note: The previous settings assume that the XILINX and LD_LIBRARY_PATH environment variables point to the appropriate areas.

Cadence/Verilog Design Flow

The following figure illustrates how Cadence Concept and Verilog-XL interact with the Xilinx Software. The design flow shows design entry, functional simulation, implementation, and timing simulation. The design is entered into Concept, using the appropriate HDL Direct library (hdl_direct_lib) and the appropriate Xilinx Concept library for the device architecture. If the design is purely schematic, it can then be passed to Verilog-XL for functional simulation after analyzing it with Concept 2XIL.

Note: The "Top Portion of Figure A-1" figure and the "Bottom Portion of Figure A-1" figure are enlarged versions of the "Cadence/ Verilog Interface Design Flow" figure.

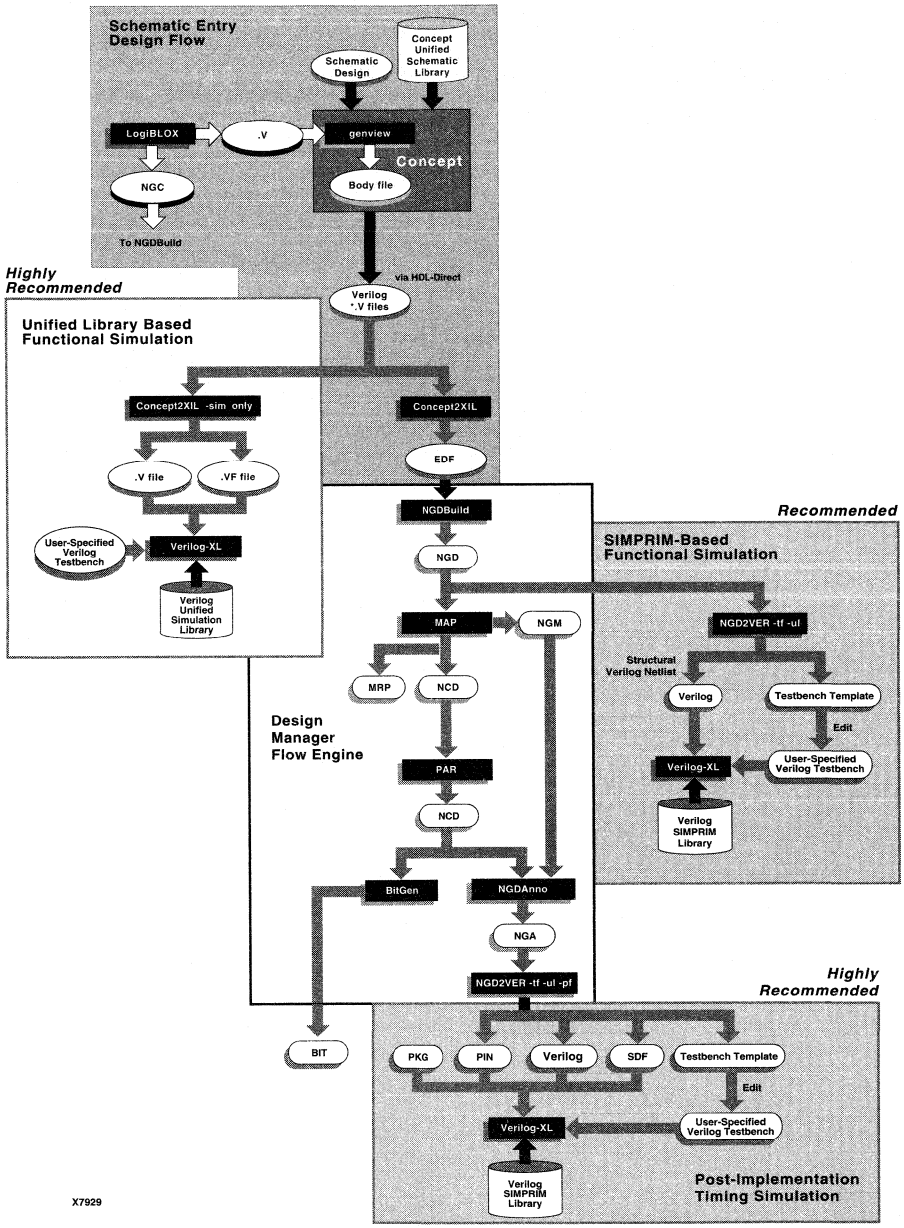


Figure A-1 Cadence/Verilog Interface Design Flow

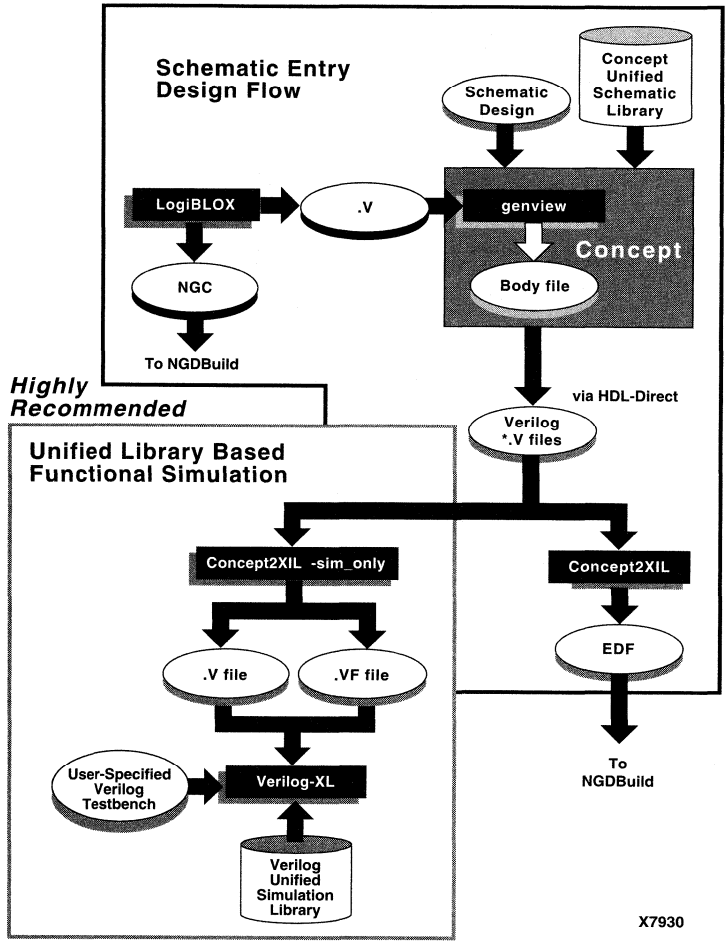


Figure A-2 Top Portion of Figure A-1

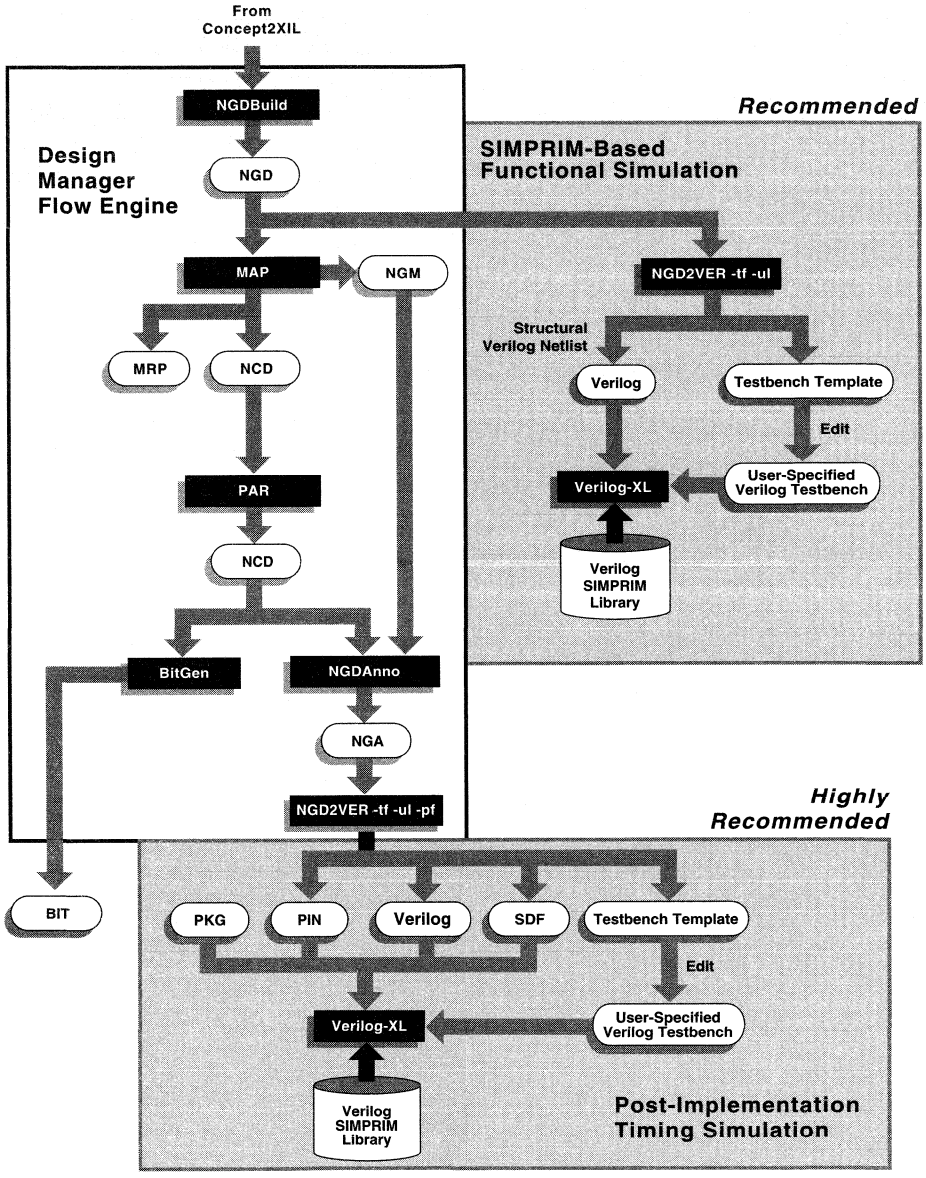


Figure A-3 Bottom Portion of Figure A-1

- Once the design is verified, the Concept schematic is processed by CONCEPT2XIL to create an EDIF (EDF) file.
- If the design contains non-schematic blocks, then functional simulation may be performed after NGDBuild is completed. All designs can be simulated at this point since non-schematic blocks, if any exist, are merged together by NGDBuild.
- The EDF file is passed to the Xilinx Alliance Series Design Implementation Tools (Design Manager Flow Engine) for implementation.
- The Xilinx implementation tools create a structural back-annotated Verilog file for timing simulation, and can optionally create a template test fixture file.
- Simulation vectors are added to a copy of the test fixture template, then the Verilog netlist and the test fixture are passed to Verilog-XL for timing simulation.

Setting Up for Concept

To set up your system for Concept, you must have the global.cmd, master.local, and cds.lib files in the design directory. Refer to the *Cadence Interface/Tutorial Guide* for more information.

Global.cmd File

You need a global.cmd file that references the proper libraries. The following is a sample global.cmd file.

```
master_library "./master.local" ;

library "xce4000x" ,

        "xcepads",

        "hdl_direct_lib",

        "standard" ;

use "my_design.wrk" ;

root_drawing "my_design" ;
```

The entries following the "library" reference are aliases to libraries from which you can access components for your design, in addition to those listed in $CDS_INST_DIR/lib/master.lib.

One of the entries in the "library" reference must point to the Xilinx family of devices you are using. In this example, the "xce4000x" alias points to the XC4000X (XC4000 EX/XL/XV) family library. The explicit path to each library is defined in *master.local*. Note the presence of "hdl_direct_lib"; this is required for HDL Direct Support. The "xcepads" library contains the Xilinx pad symbols. The "use" line points to a file (typically with a .wrk extension) that Concept can use to store references to blocks which are specific to your design. If your global.cmd file has a "use" directive specifying a .WRK file, Concept will create the specified file for you if it does not already exist (as in the case of a new design).

Master.local File

The master.local file contains the actual UNIX path to the libraries referenced in global.cmd. It does not need to contain the path to libraries that are local, or which are standard Cadence-supplied libraries specified in $CDS_INST_DIR/lib/master.lib. The following is an example master.local file for an XC4000X design.

```
file_type = master_library;

"xce4000x" '/xilinx/cadence/data/xce4000x/
xce4000x.lib';

"xcepads" '/xilinx/cadence/data/xcepads/
xcepads.lib';

end.
```

Do not use variables (such as $XILINX) in this file; absolute path names are required.

Cds.lib File

This file is required by Concept2XIL, and it must point to the location that contains the VAN (Verilog Analyzer)-compiled Verilog library files. As an example, here is a sample cds.lib file for a 4000X design.

```
define xce4000x_syn /xilinx/cadence/data/ xce4000x_syn
```

The format for entries in this file is as follows.

```
define target_tech_syn path_to_XILINX/cadence/data/target_tech_syn
```

where target_tech is spartan, spartanxl, xce3000, xce4000e, xce4000x, xce5200 or xce9000.

Using HDL Direct

The Xilinx/Cadence Interface does not support SCALD methodology for design entry. HDL Direct design methodology is required. HDL Direct must be enabled whenever a schematic sheet is saved. Putting the following commands in your startup.concept file will activate HDL Direct every time Concept is invoked.

```
set hdl_direct on
set hdl_checks on
set check_signames on
set check_net_names_hdl_ok on
set check_port_names_hdl_ok on
set check_symbol_names_hdl_ok on
set capslock_off
runopl installation_path_to_cadence/tools/fet/ concept/hdl_direct/bin/autosym
```

Note: When processing designs entered using SCALD methodology, refer to Appendix C of the *HDL Direct User Guide* (from Cadence) for complete information on converting these designs for HDL Direct compliance.

Iterated Instances Versus Size Support

The Cadence Interface and Libraries do not support the SIZE property. Iterated instances should be used instead (which essentially consist of adding a bus index to the PATH attribute of the symbol body instance). Refer to the *Cadence HDL Direct User Guide* for more information.

Starting Concept

To start the Concept editor, enter the following.

```
concept &
```

Functional Simulation

This section describes functional simulation of your designs.

Testfixture: Asserting the Global Set/Reset in a Pre-NGDBuild Unified Library Functional Simulation

In a netlist for a Spartan, SpartanXL, XC4000E/EX/XL/XV design that uses STARTUP, the Global Set and Reset ("GSR") net that leads to every flip-flop is connected to the STARTUP block implicitly. Toggling the signal that controls the GSR pin is needed to begin simulation and to simulate resetting the device. Even if your design does not utilize the STARTUP block, the GSR line should be pulsed once at the beginning of simulation to simulate the initial behavior of the device.

In the Unified Library functional simulation, if the design contains a STARTUP block, you must connect the logic that controls the GSR pin on the STARTUP block to the underlying global GSR net by using a `define directive to specify a macro called "GSR_SIGNAL."

```
`define GSR_SIGNAL
testfixture.design_instance_name.signal_on_GSR_pin
```

Here testfixture is the name of the testfixture module, design_instance_name is the instance name of the instantiated design, and signal_on_GSR_pin is the net name that sources the STARTUP GSR pin.

The signal that actually hooks up to the STARTUP GSR pin should be used. In your testfixture, you may proceed to assign values to the input pin that you use as your global reset (you do not have to assign values to the actual attached STARTUP block GSR pin signal). For example, assuming "global_reset" is the name of the port controlling the GSR pin on the STARTUP symbol.

```
module my_testfixture(input_net, output_net, global_reset);

`define GSR_SIGNAL my_testfixture.uut.signal_on_GSR_pin

my_design uut (.in(input_net), .out(output_net),
.global_reset(global_reset))

reg input_net, global_reset;

initial begin

   global_reset=1;

   #300  global_reset=0;

  //assign inputs
```

However, if the STARTUP block is not used, you must directly drive the GSR. You may again use the `define directive to define a GSR signal, even though it does not explicitly exist in the schematic or HDL code. To do this, you would define a dummy Verilog register "reg test.GSR" (assuming the testfixture module name is "test", which is a name we recommend if you want to reuse the testfixture with post-NGDBuild simulation). You then need to use the `define to hook it up to the verilog models, and drive "test.GSR" in your stimulus.

```
reg test.GSR;
`define GSR_SIGNAL test.GSR
.....
.....
initial begin
   test.GSR=1
   #300 test.GSR=0;
   //assign inputs now
```

- For the 5200 family (which has a STARTUP symbol available), there is a global reset signal called "GR," use.

  ```
  `define GR_SIGNAL testfixture.design.signal_on_GR_pin
  ```

  instead. If you are not using STARTUP, then you must define a dummy signal, as previously discussed.

- For the 9500 family, to model the global PRLD signal, use the following.

  ```
  reg PRLD
  //no 9K STARTUP, so use this dummy register value
  `define PRLD_SIGNAL test.PRLD
  .....
  .....
  initial begin
    test.PRLD=1;
    #300 test.PRLD=0;
    //assign inputs now
  ```

- For the 3000A family, use the following.

```
reg GR; //no 3K STARTUP, use this example.
`define GR_SIGNAL test.GR
```

Use a similar procedure as previously described for the 9500. (Note that GR on an XC3000A is active-Low.)

Schematic Functional Simulation

You can functionally simulate your design before translating it if the design is purely schematic (no "black boxes" in your design). Assuming that HDL Direct was activated when the schematic was saved, you should run.

concept2xil -sim_only -family *target_tech design_name*

This command creates a .V and .VF (Verilog configuration file) file in your xilinx.run directory (or optionally the directory specified with the -rundir parameter). You must create a test fixture file (.tv) manually.

An example of a complete flow is as follows.

concept2xil -sim_only -family xce4000x my_design

Go to the xilinx.run directory, and create a test fixture file in a text editor, "my_testfixture.stm," for example, then run your Verilog-XL simulation by entering

verilog +delay_mode_unit *my_testfixture*.**stim** *my_design*.**v -f** *my_design*.**vf**

For more information, refer to the *Cadence Interface/Tutorial Guide.*

Post-NGDBuild Functional Simulation

If the design has blocks that have no schematics underneath (a block of HDL code, for instance), then it is necessary to compile each non-schematic block to either an NGO, EDIF or XNF file and to simulate after the Xilinx program NGDBuild has merged all the formats into one NGD file. Briefly, the flow is as follows.

concept2xil -family *target_tech design_name*

cd xilinx.run

ngdbuild-p *part_name design_name*

ngd2ver -ul -tf *design_name*.**ngd**

verilog +delay_mode_unit *test_fixture*.**stim** *design_name*.**v**

Translating a Design to EDIF

To translate a Concept design into an EDIF file for the Xilinx implementation tools to use, enter the following command.

concept2xil -family *target_tech design_name*

For example, to target my_design to the XC4000X technology.

concept2xil -family xce4000x *my_design*

The Concept2XIL program will by default put its output in the directory xilinx.run (this will be created automatically if it does not exist). You may change this to a different directory by using the -rundir option on the Concept2xil command line.

Note: The Concept2XIL program is only compatible with the Xilinx Concept libraries (xce***) where *** = target architecture. For example of a Xilinx Concept Library name: xce9000.

Timing Simulation

After implementing your design (that is, after running MAP and PAR on your design) and generating an annotated NGA netlist (with NGDANNO), you must use NGD2VER to generate a structural Verilog netlist and SDF file (Standard Delay Format) that Verilog-XL can use.

For example, for the design "my_design", enter the following.

ngd2ver -ul -tf -pf *my_design*.**nga**

This creates a *.V*, *.SDF* and *.TV* file. The -ul option causes NGD2VER to automatically add a "uselib" directive to the *.V* file that references the Xilinx-supplied Verilog SIMPRIM libraries.

The -tf option causes NGD2VER to automatically create a test fixture template file, which is named my_design.tv. You may either edit the *.TV* file to add the appropriate stimuli, or, in most cases, you should be able to re-use your functional simulation test fixture file.

If you re-use your functional simulation test fixture file and your design contains a STARTUP block, and the GSR, GTS or both pins on

the STARTUP block are connected to a signal, you will need to make one modification to the test fixture--the GSR_SIGNAL, GTS_SIGNAL, or both) macro(s) must be commented out.

```
// `define GSR_SIGNAL test.uut.signal_on_GSR_pin.
```

If you need to integrate the design into a board level schematic, you must also specify the -pf option to NGD2VER to obtain a .pin file for XIL2CDS.

To run the simulation, type.

verilog *my_testfixture*.**stim** *my_design*.**v**

Verilog-XL automatically reads in the .sdf file, since there will be a reference to it in the .V file.

Support for Board Level Simulation

Cadence ships the program XIL2CDS to produce the chips_prt, and body file needed to integrate the Xilinx FPGA or CPLD into a Concept board level simulation.

Typical syntax is as follows.

xil2cds *routed_design* **-lwbverilog**
-use *name_of_.wrk_file* **-r** *run _directory*
-family xce4000ex -mode all

In the following example, design name is "my_design_r", *.WRK* file is design.wrk, run directory is the current directory, architecture is XC4000X, -mode option specifies that all pins on the package be represented on the design body file, and -pkg specifies the location of the package pin file.

xil2cds *my_design_r* **-lwbverilog -use** *design*.**wrk -r .**
-family xce4000x -mode all

XIL2CDS creates a body for the FPGA / CPLD called my_design_r_1.

Contact Cadence to obtain the XIL2CDS program and additional details on its operation.

Pin Locking

You may place the PADs on specific pins of your target device by adding the "LOC" property to the IBUF or OBUF that connects to it.

If you use a "bussed" I/O buffer symbol (for example, IBUF8), you must add the pin constraints to the UCF file instead.

Note: You cannot put the LOC property on the PAD or the net between the PAD and I/O buffer. If you do, it will be ignored since Concept does not support properties on pads.

To add a LOC property:

1. Enter the "Attribute" mode, and select the IBUF/OBUF you wish to constrain.
2. Select Add, and enter LOC in the Name field, and the pin name in the Value field.

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

   Valid pin syntax for the grid array packages (BGA, PGA) is RC, where R is the actual row and C is the actual column of the device pin.
3. Select Done. You may re-position the LOC property above the PAD, using the Move command in Concept.

Timing Constraints

Timing constraints may be placed as properties on a TIMESPEC symbol in the design. Click on the "Attribute" button in Concept, then select the TIMESPEC symbol to display the list of properties. Select "Add" to add a new property. The Timespec label (the label that begins with "TS") is entered in the Name field, while the timing specification (for example, "FROM:FFS:TO:FFS=30ns") is entered in the Value field. By default, you may only use "TS" labels "TS01" through "TS10" with Concept. If you wish to use other labels, you must copy the $XILINX/cadence/data/xilinx.pff file to your design directory, and add entries for other labels. For more information on this subject, refer to the *Cadence Interface/Tutorial Guide.* For more information on timing constraints, see the *Development System Reference Guide.*

Appendix B

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 the Foundation Express 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 Quick Start Guide 1.5*.

FPGA Express is a Verilog/VHDL compiler designed to work with Windows 95 and Windows NT v4.0. FPGA Express can process either Verilog or VHDL files. FPGA Express writes out XNF that is fully compatible with Alliance Series Design Implementation tools. Only the 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 Release Notes.
- 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 that 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.

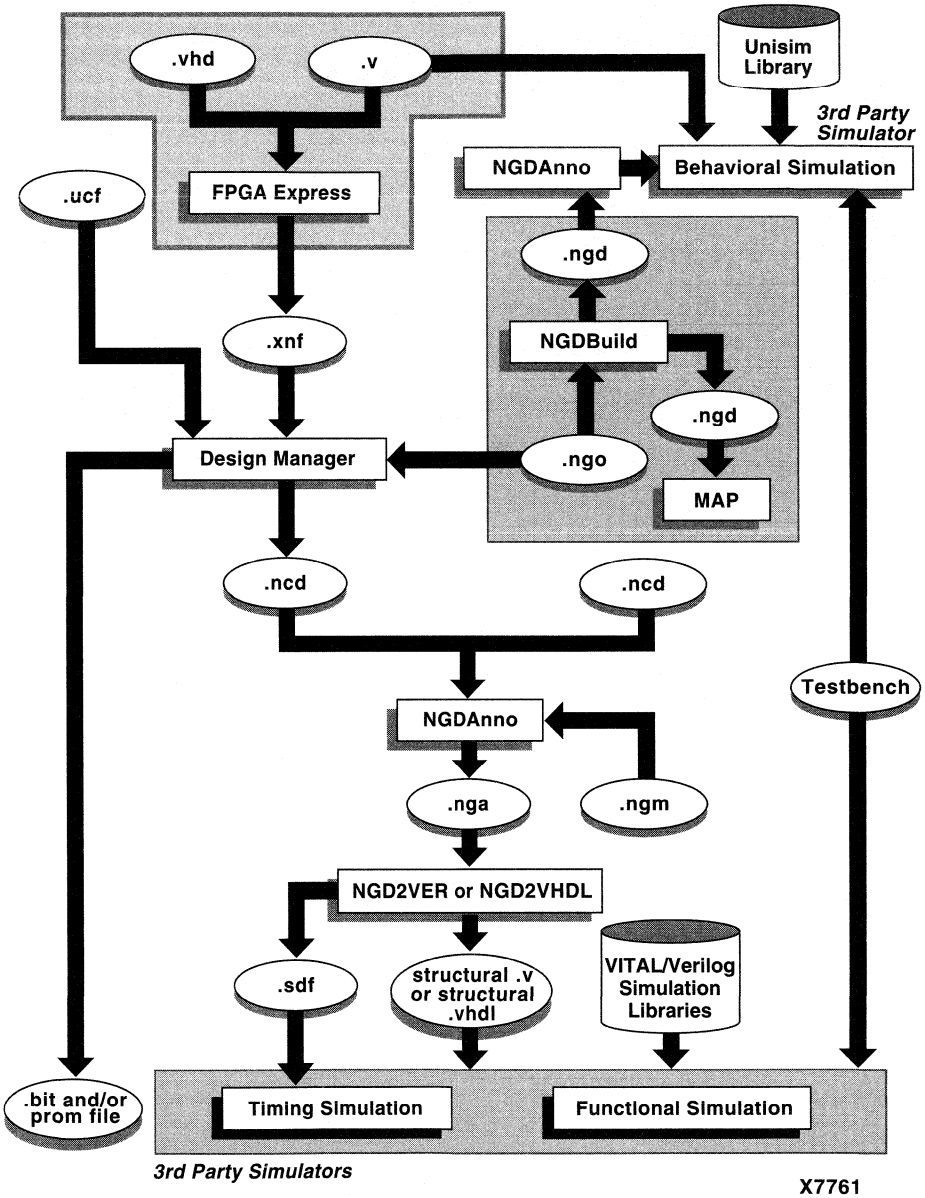


Figure B-1 Alliance FPGA Express/Xilinx Design Flow

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.

   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.

   Synthesis → Create Implementation

7. Optimize your design by selecting.

   Synthesis → Optimize Chip

8. Write an XNF file by selecting.

   Synthesis → Export Netlist

Verilog or VHDL designs are the input files for the FPGA Express design flow, and the output is an XNF file, 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.

FPGA Express synthesizes your designs based on die size. If necessary, FPGA Express can wrap carry logic from one column to the next. You should use the device that is targeted for synthesis by FPGA Express as the target device for the Xilinx place and route tools.

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 a general overview of Xilinx simulation, refer the *Development System User Guide*. For information on using the Design Manager in HDL simulation, the *Design Manager/Flow Engine Reference/User 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. 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 *"Synopsys (XSI) Interface Notes" appendix*.

Using LogiBLOX with FPGA Express

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

Appendix C

Mentor Graphics Interface Notes

This appendix describes how to set up the Mentor Graphics interface and associated libraries. You are guided through 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/Tutorial Guide* is available online and viewable with a DynaText browser.
- 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.

- Alliance 1.5 *Release Documentation* describes installation setup and current issues regarding the use of the Mentor Graphics Interface.

Setting Up the Xilinx/Mentor Interface

In addition to the environment variables discussed in the "Installing the Software" chapter, 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
```

The following is an example of how to set the 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 "Installing the Software" chapter.

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.

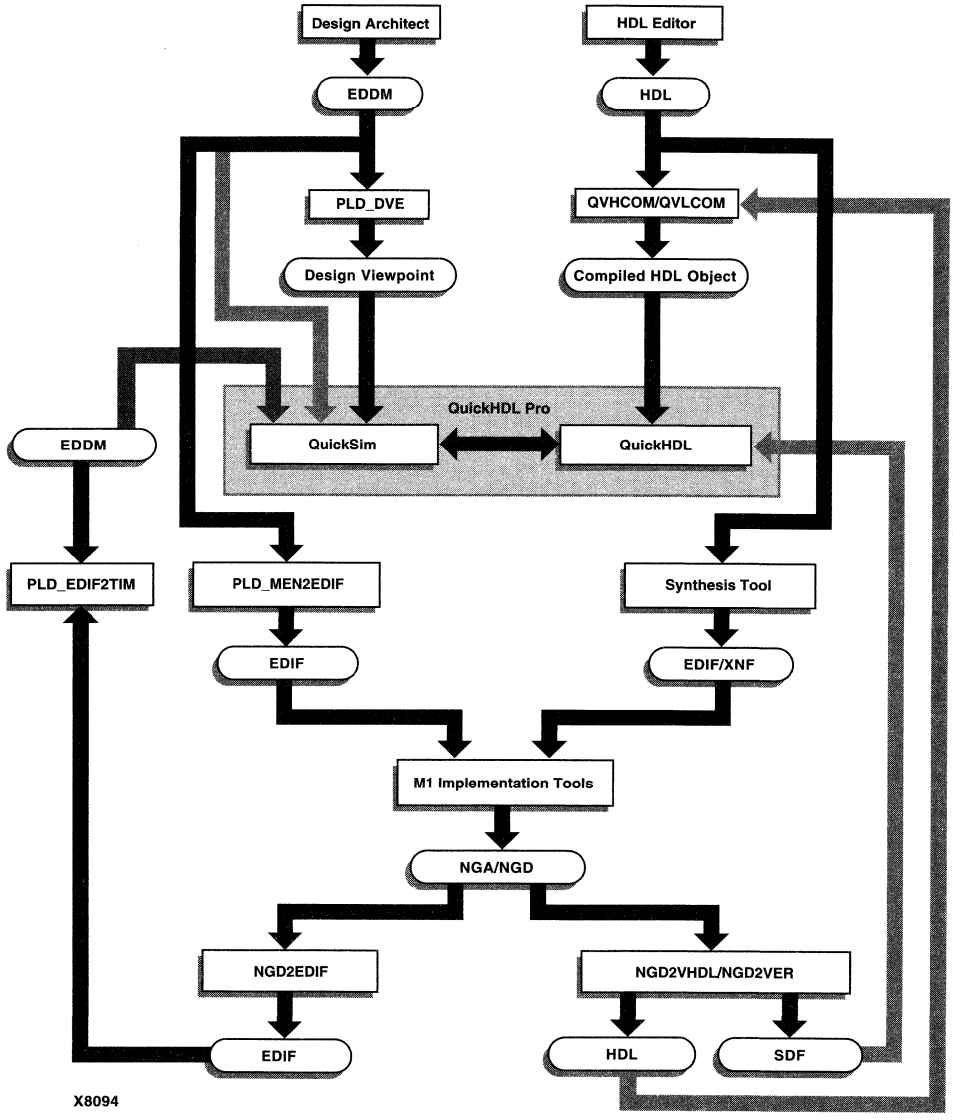


Figure C-1 Mentor/Xilinx Software Flow

- 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/Tutorial 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/ Tutorial 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/Tutorial 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/Tutorial 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/Tutorial 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 D

Synopsys (XSI) Interface Notes

This appendix provides information on setting up the Synopsys interface 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"
- "Synopsys Interface Design Flow"
- "Examples of Synopsys Setup Files"
- "Timing Constraints and DC2NCF"
- "Using FPGA Compiler"
- "Entity Coding Examples"

Documentation

The following documentation is available for the Synopsys interface.

- The *Synopsys (XSI) Interface/Tutorial Guide* is available on the CD-ROM included with your software package.
- Alliance 1.5 *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 XACTstep vM1.X.X.*

Setting Up the Synopsys Interface

In addition to the environment variables described in the "Installing the Software" chapter, 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 Compiler v1997.01or a later version, you must re-compile the Xilinx DesignWare (XDW) libraries.

The XSI (Xilinx Synopsys Interface) simulation and XDW (Xilinx DesignWare) libraries are compiled for Synopsys v1997.01. If you are using the latest version of XSI with a version of Synopsys newer than v1997.01, 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.

You can compile the libraries in the $XILINX area, or in some other area. If you compile the libraries in the $XILINX area, you must have write permissions to this area. If you copy the $XILINX/synopsys directory to a local area, you can re-compile the libraries without needing write permissions for the $XILINX area. 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 following.
   - 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. Enter the following.

   ```
   ./analyze.csh
   ```

   Go back to the $XILINX/synopsys/libraries/sim/src directory.
5. Go to the simprims directory. Enter the following.

   ```
   ./analyze.csh
   ```

   Go back to the $XILINX/synopsys/libraries/sim/src directory.
6. Go to the unisims directory. Enter the following.

   ```
   ./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. Enter the following command.

   ```
   ./analyze.csh
   ```

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

8. Go to the xc9000/ftgs directory. Enter the following command.

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

Synopsys Interface Design Flow

The "XSI Design Flow" figure shows the steps in the XSI design flow.

Note: For more information on the XSI design flow, see the *Synopsys (XSI) Interface/Tutorial Guide*.

Design Flow Input

The following are inputs to the Design Compiler and the FPGA Compiler.

- Synthesis script (DC and FPGAC script in the following figure)
- Source HDL (Verilog or VHDL)
- .synopsys_dc.setup (Synopsys setup file)

Design Flow Output

The following are outputs from the compilers.

- *Design_name*.dc

  Timing constraints written by both compilers. This file is used as input to DC2NCF to create a netlist constraints file *design*.ncf.

- *Design_name*.edif

  Design for Virtex written by the FPGA Compiler in EDIF format.

- *Design_name*.sxnf

  Design netlist written by the FPGA Compiler in XNF format.

- *Design_name*.sedif

  Design netlist written by the Design Compiler in EDIF format.

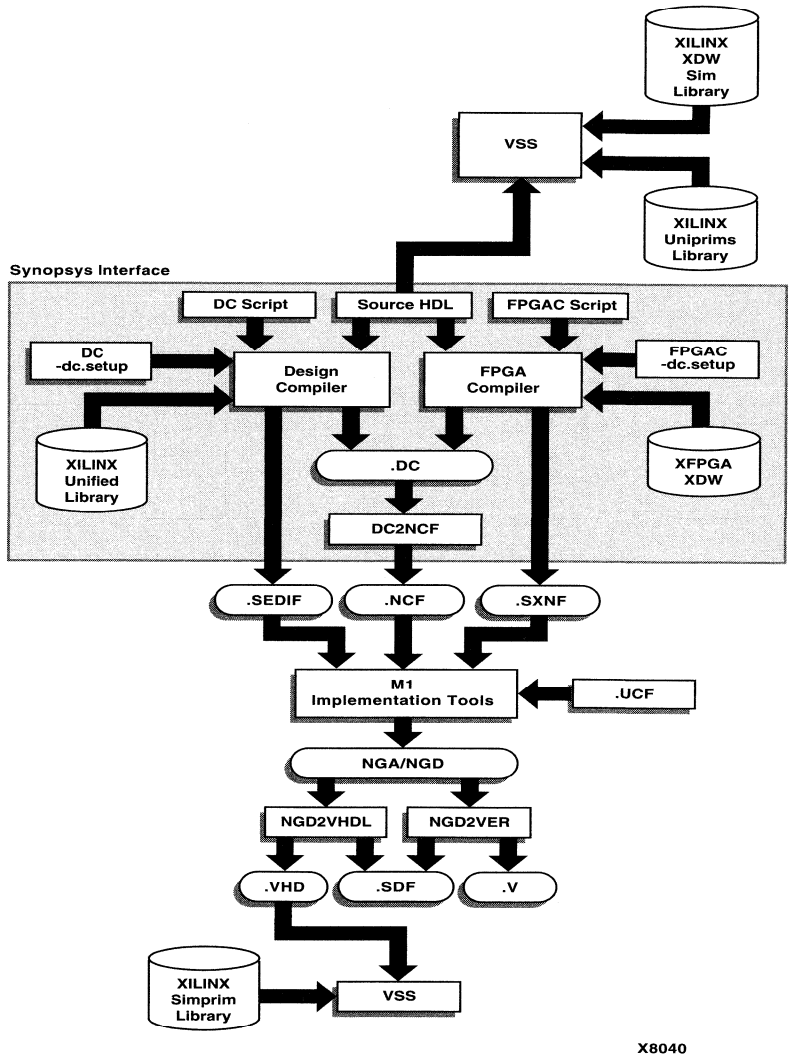


Figure D-1 XSI Design Flow

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.  */
/* ================================================ */
```

Timing Constraints and DC2NCF

Timing constraints issued to Synopsys to control the synthesis process can be carried forward to the design implementation tools to control the place and route process. It is important that the constraints you apply to both the synthesis and place and route processes are both realistic and achievable.

The Xilinx DC2NCF utility converts the timing constraints applied to your design in the Synopsys environment to equivalent constraints that control the Xilinx place and route process. The automatic translation of these constraints is convenient because you do not need to

apply the constraints twice. In addition, it ensures that the constraints used by the Xilinx tools are equivalent to those applied in Synopsys.

DC2NCF supports translation of the following timing constraints.

- create_clock
- set_input_delay
- set_output_delay
- set_max_delay
- set_false_path

If there are unsupported timing constraints in the Synopsys script file, DC2NCF issues a warning message and the constraints are not translated.

DC2NCF Design Flow

You should validate your design timing constraints by first constraining your design and then compiling it. After compiling your design (for XC4000E/EX/XL/XV/XLA FPGA designs, you must also run the replace_fpga command), write your design as a netlist and a corresponding script file that contains the constraints.

Warning: Always generate timing constraints script files with the Synopsys dc_shell write_script command or the Design Analyzer File, Save Info, Design Setup command sequence.

Using FPGA Compiler

Before writing either the netlist or the constraints file, any hierarchy in your XC4000E/EX/XL/XV/XLA designs must be flattened. Flattening your design removes hierarchy information from the Synopsys internal database. The hierarchical net-names and instance-names assigned to objects during compilation are retained and written into the output netlist.

Note: The downstream Xilinx tools will reconstruct most of the design's hierarchy from the information carried in the instance-names and net-names.

To flatten the design's hierarchy prior to writing a netlist and constraints file, use the following Synopsys command.

Note: Do not use the following command for Virtex designs.

`ungroup -flatten -all`

To write-out the design's netlist in XNF format, use the Synopsys command.

`write -format xnf -output` *output_file_name*`.sxnf`

To write-out the design's constraints as a Synopsys script file, use the command.

`write_script >` *output_file_name*`.dc`

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.

FPGA Compiler/Design Compiler and LogiBLOX

For information on using LogiBLOX in the XSI flow, refer to the *Synopsys (XSI) Interface/Tutorial Guide*.

Appendix E

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"
- "Viewlogic Interface Design Flow"
- "Setting Up Project Libraries"
- "Assigning a Pin Location"

Documentation

The following documentation is available for the Viewlogic interface.

- *Viewlogic Interface/Tutorial Guide* is available on-line and viewable with the DynaText browser.
- Xilinx *Release Document* describes installation setup and current issues regarding the use of the Viewlogic interface, and is available on the supplied CD-ROM.

Setting Up Viewlogic Interface on Workstations

In addition to the environment variables described in the "Installing the Software" chapter, 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

In addition to the environment variables described in the "Installing the Software" chapter, 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 FLEXlm™ 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.

Viewlogic Interface Design Flow

The following figure shows the Viewlogic interface design flow.

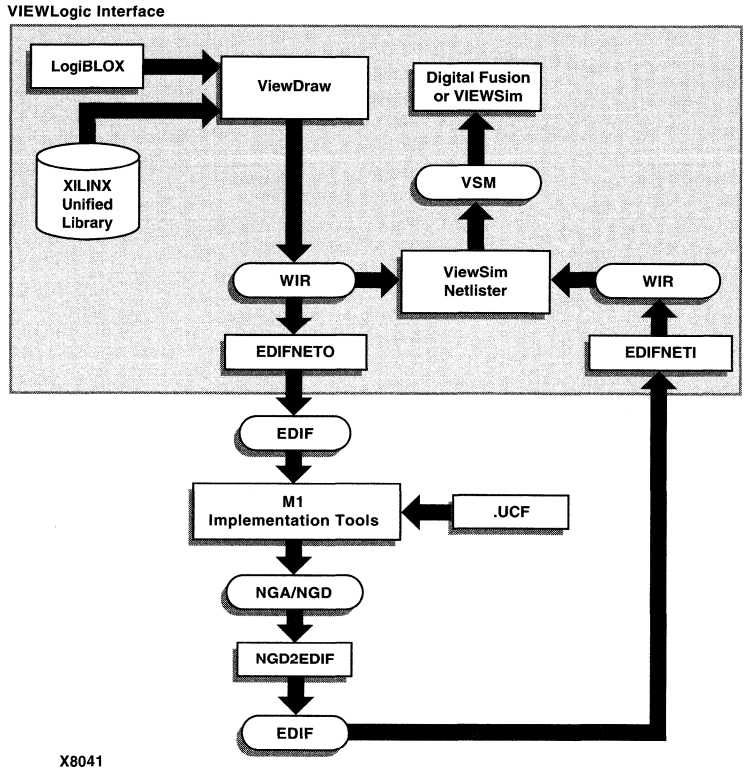


Figure E-1 Viewlogic/Xilinx Software Design Flow

Steps in the Design Flow

The Viewlogic Design Flow includes the following steps.

- The design flow starts with ViewDraw® using the Xilinx Unified Library components and (optionally) LogiBLOX components.
- When the schematics are saved, WIR files are created. EDIFNETO translates these WIR files to EDIF 2 0 0 format to be passed to the Xilinx M1 implementation tools for implementation.

Note: The option *Xilinx* must be entered as the level name for EDIFNETO.

- A ViewSim® netlist file (.VSM) must be created for simulation. This file is created from WIR files that have come directly from the Viewlogic design entry tools, or from the Xilinx Alliance Series Design Implementation Tools via EDIFNETI. Only an EDIF file from the placed and routed design provides full timing information for your design.
- The ViewSim netlist file is then loaded into the Viewlogic simulator for simulation.
- Digital Fusion, the Viewlogic simulation tool with VHDL simulation capabilities, is required only for functional simulation of designs containing LogiBLOX components with VHDL models; ViewSim, Viewlogic's gate-level simulator, will accept VSM files for any other Xilinx simulations.

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)
```

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.

Initializing Xilinx Commands in ViewDraw

After the environment variables are set and the libraries are defined, you can start your schematic design. In the Viewlogic interface, there are five new commands in the Tools menu. You must initialize these commands in MS-DOS® as shown in the following table.

To initialize the two LogiBLOX commands, enter the following.

`custmenu` *xilinx-path*`\viewlog\data\viewblox.txt`

To initialize the three Xilinx flow commands, enter the appropriate command for your version of Windows.

For Windows 95, enter the following command.

`custmenu` *xilinx-path*`\viewlog\data\xvdraw95.txt`

Windows NT, enter the following command.

`custmenu` *xilinx-path*`\viewlog\data\xvdrawnt.txt`

You should see the following five commands in the ViewDraw Tools menu.

Table 4-1 Xilinx ViewDraw Commands

Command	Function
Add LogiBLOX	Use to create LogiBLOX components
Change LogiBLOX	Use to modify an existing LogiBLOX component
Write Xilinx EDIF	Use to create the EDIF netlist to pass to the Design Manager
Xilinx Functional Simulation	Use to prepare a design with uncompiled elements for a functional simulation
Read Xilinx Timing EDIF	Use to transfer the annotated EDIF to Workview Office for a timing simulation

Note: See the *Viewlogic Interface/Tutorial Guide* for more information on these commands. Also, refer to the "Using LogiBLOX with CAE Interfaces" appendix for more information on LogiBLOX.

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/ Tutorial Guide.* For more information on timing constraints, see the *Development System Reference Guide.*

Appendix F

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.

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 Reference/User Guide* is available on the CD-ROM supplied with your software and viewable with the DynaText browser.
- The LogiBLOX online help can be accessed from LogiBLOX, GUI.
- Alliance 1.5 *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 "Installing the Software" chapter 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 "Installing the Software" chapter manual and in the appropriate appendix in this manual.

Viewlogic Interface Environment Variables

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 LogiBLOX 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 G

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, Synopsys (XSI) Interface/ Tutorial 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 start-up 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 Verilog GTS/ GSR simulation see the *Cadence Interface/Tutorial Guide.* For VHDL

designs, use components in the following table and refer to the *Development System User Guide* for HDL simulation information.

Table G-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 G-2 STARTBUF Library Component

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

a. For 5200, GSR pin is GR.

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 G-3 BSCAN Library Components

Name	Family	Description	Outputs	Inputs
BSCAN	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, 5200[a]	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	—

a. 5200 has three additional output pins: Reset, Update, 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 G-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 G-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

Table G-5 RAM and ROM Library Components

Name	Family	Description	Outputs	Inputs
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
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 G-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[a]	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[b]	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

a. BUFGP_F for Synopsys when connected to dedicated Pad

b. BUFGS_F for Synopsys when connected to dedicated Pad

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 G-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 G-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 G-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 G-8 IOB Components

Name	Family	Description	Outputs	Inputs
)FDX	4000E/L, 4000EX, 4000XL, 4000XV, Spartan, Virtex	Single output D flip-flop with clock enable	Q	D, C, CE
)FDTX	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
LD_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 G-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 H

Alliance Constraints

This appendix describes the steps used to implement a design with the Xilinx tools with a specific focus on how constraints are entered at each stage of design processing. It also describes the more common constraints you can apply to your designs to control the timing and layout of FPGAs or CPLDs. For a complete listing of all supported constraints, refer to the *Libraries Guide*. For a more complete description of timing constraints, refer to the *Development System Reference Guide*. This appendix includes the following sections.

- "Entering Design Constraints"
- "Translating and Merging Logical Designs"
- "Constraints File Overview"
- "Timing Constraints"
- "Layout Constraints"
- "Efficient Use of Timespecs and Layout Constraints"
- "Standard Block Delay Symbols"
- "Table of Supported Constraints"
- "UCF Syntax Examples"
- "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.

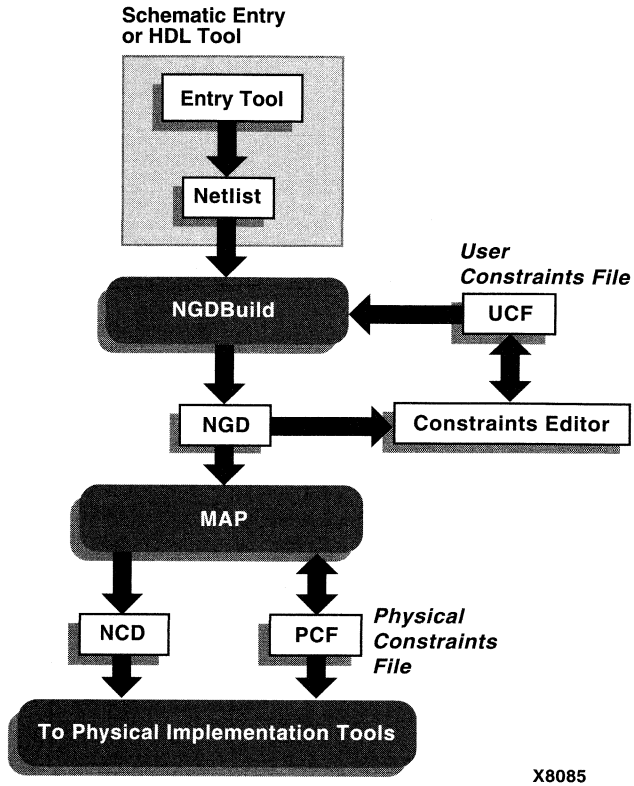


Figure H-1 Entering Design Constraints

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).

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.

Constraints File Overview

The following sections summarize the functions of user constraints files.

Netlist Constraints File (NCF)

The Netlist Constraints File was developed as an alternative means for third party vendors to provide the Xilinx tools with design constraints. Historically, these constraints are in the design netlist and are annotated to the equivalent elements within the designs NGO file. In the current release of the Xilinx software, translating a logic design netlist to a NGO file includes annotating constraints present in the NCF file to the NGO design elements. The NCF file is local in scope and therefore must have the same root name as the netlist being translated. In addition, the local scope of the netlist allow entries within the NCF file to refer to specific nets or symbols without prefixing the entire hierarchical path of the net or symbol.

The "Entering Design Constraints" figure illustrates the Synopsys workstation design flow which utilizes a NCF file. The program DC2NCF converts the Synopsys constraints into NCF syntax. Refer to the *Synopsys (XSI) Interface /Tutorial Guide* for detailed information on using this flow.

Note: Whether or not an NGO is written to disk for a LogiBlox component is dependent on the specific design creation technology being used. Many synthesis flows require that an NGO file be written to disk during component specification, while the typical schematic flow provides for on-the-fly compilation of NGO files.

User Constraints File

The User Constraints File (UCF) provides a convenient mechanism for constraining a logical design without returning to the design entry tools. The process of building the complete logical design representation (NGD files) is performed by NGDBuild. In developing this

complete design database, NGDBuild annotates design constraints in the UCF file. The syntax for the UCF is identical to the syntax for the NCF.

One main difference between the NCF and the UCF is how objects in the files are identified. A constraint applied with the UCF file must specify the complete hierarchical path name for the constrained instance or net, while an NCF constraint must only reference the specific net or symbol within the associated netlist. The difference in hierarchical path name requirements arises from the scope of the design files themselves; NCF files have a local scope and can therefore tolerate local references, while UCF files have a global scope and therefore require full hierarchical path names.

Another difference is the UCF can override NCF constraints. Because the NCF is considered an alternative mechanism for design creation tools to pass constraints to the Xilinx implementation tools, no conflicts should occur. In contrast, the UCF annotation includes a resolution mechanism to allow UCF constraints to write over constraints in the NCF. UCF constraints are more important because they occur later in the design flow, and provide a mechanism for establishing or modifying logical design constraints without requiring the user to re-enter a schematic or synthesis tool.

Note: Versions prior to M1.2 required the -uc switch to identify a UCF which needs to be annotated to the design. Versions M1.2 and later allow UCF file annotation to be performed by default if the UCF file has the same base name as the input.

Physical Constraints File

The NCD, or physical representation of the design, describes the design in terms of FPGA resources. The layout and timing analysis tools work with the NCD representation. The Physical Constraints File (PCF) contains the constraints as they relate to the NCD. Layout and timing constraints are written in terms of the physical design's components (COMPs), fractions of COMPs (BELs) and collections of COMPs (macros). Because of this different design viewpoint, the PCF syntax is not necessarily the same as the logical design constraints files (UCF/NCF). Furthermore, because the PCF file is written for use by the physical implementation tools, logical names may no longer be similarly represented in the physical design.

If you modify the PCF file, enter your constraints after the "SCHEMATIC END;" line. Otherwise, your constraints are overwritten every time MAP is executed using that PCF file.

Case Sensitivity

Xilinx constraints are case sensitive because EDIF is a case-sensitive format. You should specify the net names and instance names exactly as they are in your schematic or code. Also, be consistent when using TNMs and other user-defined names in your constraints file; always use the same case throughout. For site names (such as "CLB_R2C8" or "P2"), use upper-case letters only.

Timing Constraints

The following sections summarize the functions of timing constraints. The PERIOD and OFFSET constraints are preferred for Xilinx designs.

PERIOD Constraint

The following schematic example illustrates the use of the PERIOD constraint referenced to timegroups *CLK2_GRP* and *CLK3*. This example is for multiple clock designs. Use FROM:TO to constrain data paths between the two clock domains.

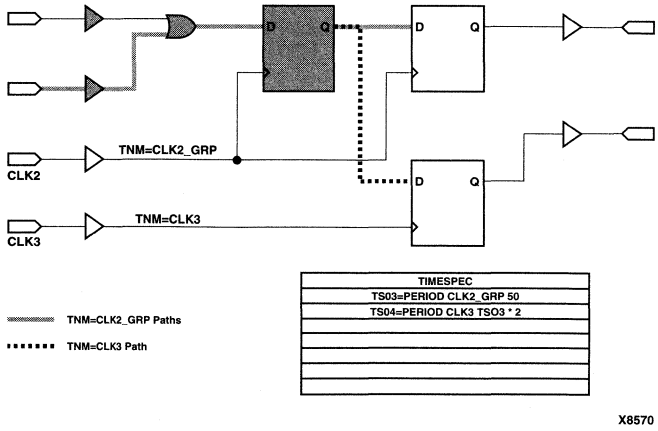


Figure H-2 PERIOD Example

Following is the corresponding UCF file.

```
# UCF PERIOD style Timespecs

  NET CLK2 TNM = CLK2_GRP ;
  NET CLK3 TNM = CLK3 ;

  TIMESPEC TS03 = PERIOD CLK2_GRP 50 ;
  TIMESPEC TS04 = PERIOD CLK3 TS03 * 2 ;
```

In addition, the example also shows how constraints and nets may be given the same name because they occupy separate name-spaces. Also, it shows the constraint syntax whereby one Timespec is defined relative to another (the value of TS04 is declared to be two times that of TS03).

The PERIOD constraint covers all timing paths which start or end at a register, latch, or synchronous RAM which is clocked by the referenced net. The only exception to this rule are paths to output pads which are not covered by the PERIOD constraint. (Input pads, which are the source of a "pad-to-setup" timing path for one of the specified synchronous elements, are covered by the PERIOD constraint.)

The TIMESPEC form of the PERIOD constraint allows flexibility in group definitions and allows you to define clock timing relative to another TIMESPEC. The flexibility of the TIMESPEC form of the PERIOD constraint arises from being able to modify the contents of the TIMEGRP once the design has been mapped. By adding or removing objects from the TIMGRP, which are listed in the PCF file, the paths which are covered by the PERIOD constraint may be altered.

If the flexibility of the TIMESPEC form is not required, the NET form of the PERIOD constraint may be used. The syntax for the NET form of the PERIOD constraint is simpler than the TIMESPEC form, while continuing to provide the same path coverage. The following example illustrates the syntax of the NET form of the PERIOD constraint.

```
# NET form of the PERIOD timing
# constraints (no TSIdentifier)

  NET CLK PERIOD = 40 ;
```

This is the recommendation of using PERIOD on a single clock design in which data does not pass between the clock domains.

PERIOD includes clock skew in the path analysis.

OFFSET Constraint

Offsets are used to define the timing relationship between an external clock and its associated data-in or data-out-pin. Using this option allows you to:

- Calculate whether a setup time is being violated at a flip-flop whose data and clock inputs are derived from external nets.
- Specify the delay of an external output net derived from the Q output of an internal flip-flop being clocked from an external device pin.

The three types of offset specifications are global, specific, and group. Because the global and group OFFSET constraints are not associated with a single data net or component, these two types can also be entered on a TIMESPEC symbol in the design netlist with Ts*id*. See the timing constraints chapter in the *Development System Reference Guide* for details.

In the following example, the OFFSET constraint is applied to a net connecting with a PAD (see "Using OFFSET Constraints" figure). It defines the delay of a signal relative to a clock, and is only valid for registered data paths. The OFFSET constraint specifies the signal delay external to the chip, allowing the implementation tools to automatically adjust relevant internal delays (CLK buffer and distribution delays) to accommodate the external delay specified with this constraint.

```
# Net form of the OFFSET timing constraint
  NET ADD0_IN OFFSET = IN 14 AFTER CLK_IN;
```

In analyzing OFFSET paths, the Xilinx timing tools adjust the PERIOD associated with the constrained synchronous element based on both the timing specified in the OFFSET constraint and the delay of the referenced clock signal.

In the following figure, assume a delay of 8 ns for the signal CLK to arrive at the CLB, a 5 ns setup time for ADD0, and a 14 ns OFFSET delay for the signal ADD0. Assume a period of 40 ns is specified. The Xilinx tools allocate 29 ns for the signal ADD0 to arrive at the CLB input pin (40ns - 14ns + 8ns - 5ns = 29ns).

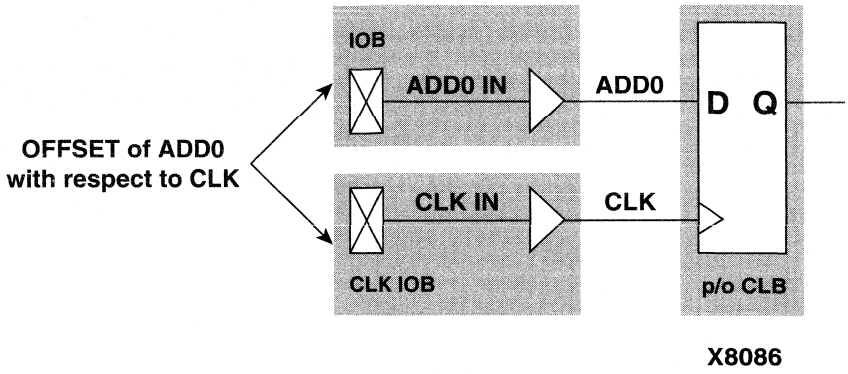


Figure H-3 Using OFFSET Constraints

This same timing constraint may be applied using the FROM PADS TO FFS timing constraint. However, using a FROM TO methodology would require you to know the intrinsic CLK net delay, and you would have to adjust the value assigned to the FROM TO Timespec. The internal CLK net delay is implicit in the OFFSET / PERIOD constraint. Furthermore, migrating the design to another speed grade or device will require modification of the FROM TO Timespec to accommodate the new intrinsic CLK net delay. It should be noted that an alternative solution is to use the flip-flop in the IOB of certain FPGA architectures (XC4000E / EX, for instance), as the clock-to-setup time is specified in the *Programmable Logic Data Book*.

Note: Relative Timespecs can only be applied to similar Timespecs. For example, a PERIOD Timespec may be defined in terms of another PERIOD Timespec, but not a FROM TO Timespec.

From:To Constraint

When using the From:To constraint, the path(s) that are constrained are specified by declaring the start point and end point, which must be a pad, flip-flop, latch, RAM, or user-specified sync point (see TPSYNC). To group a set of endpoints together, you may attach a TNM attribute to the object being an instance or macro (or to a net that is an input to the object). With a macro the TNM traverses the hierarchy to tag all relevant objects. A TIMEGRP is a mechanism for combining two or more sets of TNMs or other TIMEGRPs together, or

alternatively, to create a new group by pattern matching (grouping a set of objects that all have output nets that begin with a given string).

TNMs are used by the current Xilinx tools in the same way as the XACT*step* 5.2 tools—identification of a group of design objects which are to be referenced within a Timespec. If a TNM is placed on a net, the tools determine TNM membership by tracing forward from the specified net to all the valid endpoints of the net. Refer to the *Development System Reference Guide* for more information on this subject.

The following schematic shows an example of TNM, TIMESPEC, and TIMEGRP statements.

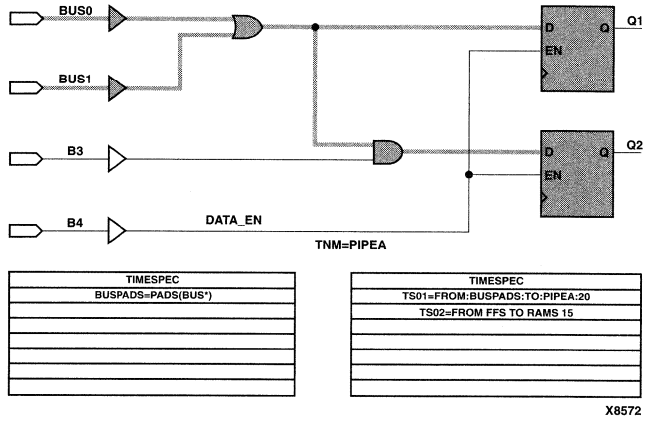


Figure H-4 Example TNM, TIMESPEC, and TIMEGRP

Following is the UCF file that corresponds to the preceding figure:

```
# This is a comment line
# UCF FROM : TO style Timespecs

  NET DATA_EN TNM = PIPEA ;
  TIMEGRP BUSPADS = PADS(BUS*) ;
  TIMESPEC TS01 = FROM BUSPADS TO PIPEA 20 ;

# Spaces or colons (:) may be used as field separators

  TIMESPEC TS02 = FROM FFS TO RAMS 15 ;
```

The first line of the previous example illustrates the application of the TNM (Timing Name) *PIPEA* to the net named *DATA_EN*. The second line illustrates the TIMEGRP design object formed using a pattern matching mechanism in conjunction with the pre-defined TIMEGRP

"PADS". In this example, the TIMEGRP named *BUSPADS* will include only those PADs with names that start with *BUS*.

Each of the user-defined Timegroups is then used to define the object space constrained by the timing specification (Timespec) named TS01. This timing specification states that all paths from each member of the *BUSPADS* group to each member of the *PIPEA* group needs to have a path delay that does not exceed 20 nanoseconds (ns are the default units for time). The TIMESPEC TS02 constraint illustrates a similar type of timing constraint using the predefined groups FFS and RAMS.

All From:To Timespecs must be relative to a Timegroup. The previous example illustrates that Timegroups may be defined by the user either explicitly (TIMEGRPs) or implicitly (TNMs), or they may be predefined groups (PADS, LATCHES, FFS, RAMS).

There is an additional keyword that can be added to the From:To spec that allows the user to narrow the set of paths that are covered. By using the From Thru To form, you may constrain only those paths that go through a certain set of nets, defined by the TPTHRU keyword, as shown in the following example.

```
# UCF FROM:TO Timespec using THRU

NET $1I6/thisnet TPTHRU=these ;
NET $1I6/thatnet TPTHRU=these ;

TIMEGRP sflops=FFS(DATA*) ;
TIMEGRP dflops=FFS(OUTREG*) ;

TIMESPEC TS23=FROM:sflops:THRU:these:TO:dflops:20 ;
```

Here, only those paths that go from the Q pin of the *sflops* through the nets $1I6/thisnet and $1I6/thatnet and on to the D pin of *dflops* will be controlled by TS23.

TPSYNC Attribute

In the XACT 5.x/6.x methodology, only flip-flops, RAMs, latches and pads could be identified as a startpoint or endpoint for a timing specification. However, in the Xilinx toolset, you can now define any node as a source or destination for a Timespec with the TPSYNC keyword. TPSYNC is attached to a set of nets, pins, or instances in the design.

For instance, suppose a design has a PAD ENABLE_BUS that must arrive at the enable pin of several different tristate buffers in less than

a specified time. With the current Xilinx tools, you can now define the tristate buffer as an endpoint for a timing spec.

The following figure illustrates TPSYNC.

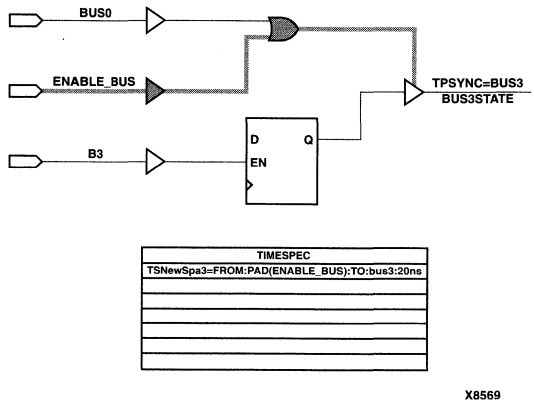


Figure H-5 TPSYNC Example

Following is the corresponding UCF file.

```
# TPSYNC example; pad to a 3-state buffer enable pin
# Note TPSYNC attached to 3-state buffer's output NET

NET BUS3STATE TPSYNC=bus3;
TIMESPEC TSNewSpc3=FROM:PAD(ENABLE_BUS):TO:bus3:20ns;
```

In the NET statement shown previously, the TPSYNC is attached to the output net of a tristate buffer called *BUS3STATE*. If a TPSYNC is attached to a net, then the source of the net is considered to be the endpoint (in this case, the tristate buffer itself). The subsequent TIMESPEC statement can use the TPSYNC name just as it uses a TNM name.

The next TPSYNC example shows how you may use the keyword PIN instead of NET if you wish to attach an attribute to a pin.

```
# Note TPSYNC attached to 3-state buffer's enable PIN

PIN $116/BUSMACRO1/TRIBUF34.T TPSYNC=bus1;
TIMESPEC TSNewSpc1=FROM PAD(ENABLE_BUS) TO bus1 20ns;
```

In this example, the instance name of the tristate buffer is given followed by the pin name of the enable (.T). If a TPSYNC is attached to a primitive input pin, then the primitive's input is considered the

startpoint or endpoint for a timing specification. If it is attached to a output pin, then the output of the primitive is used.

The last TPSYNC example shows how you may use the keyword INST if you wish to attach an attribute to a instance.

```
# Note TPSYNC attached to 3-state buffer INSTANCE (UCF
# file)

INST $1I6/BUSMACRO2/BUFFER_2 TPSYNC=bus2;
TIMESPEC TSNewSpc2=FROM:PAD(ENABLE_BUS):TO:bus2:20ns;
```

If a TPSYNC is attached to an instance, then the output of the instance is considered the startpoint or endpoint for a timing specification.

Ignoring Paths

When a Timespec is declared that includes paths where the timing is not important, the tools may create a less optimal route since there is more competition for routing resources. This problem can be alleviated by using a TIG (timing ignore) attribute on the non-critical nets. TIG will cause all paths that fan out from the net or pin where it is applied to be ignored for purposes of timing analysis.

```
#TIG example (UCF file)

  NET $1I456/slow_net TIG=TS01, TS04 ;
```

The previous syntax indicates that `$1I456/slow_net` should not have the Timespec TS01 or TS04 applied to it.

On the other hand, the following syntax indicates that the layout tools should ignore paths through the `$1I456/slow_net` net for *all* known Timespecs.

```
#Global TIG example

  NET $1I456/slow_net TIG;
```

Controlling Skew

Skew is the difference between the minimum and maximum of the maximum load delays on a net. You can control the maximum allowable skew on a net by attaching the MAXSKEW attribute directly to the net.

```
#MAXSKEW example

  NET $1I345/net_a MAXSKEW=3 ;
```

The above indicates that 3 ns is the maximum skew allowed on `$1I345/net_a`. For a detailed example of how MAXSKEW works, see the *Development System Reference Guide*.

Constraint Precedence

A user may assign a precedence to Timespecs only within a certain class of constraints. For example, you may specify a priority for a particular FROM TO specification to be greater than another, but may not specify a FROM TO constraint to have priority over a TIG constraint.

The following example illustrates the explicit assignment of priorities between two same-class timing constraints, the lowest number having the highest priority.

```
# Priority UCF Example

TIMESPEC TS01 = FROM GROUPA TO GROUPB 40 PRIORITY 4;
TIMESPEC TS02 = FROM GROUP1 TO GROUP2 35 PRIORITY 2;
```

The following two subsections describe the order of precedence for constraint files and timing constraints.

Layout constraints also have an inherent precedence which is based on the type of constraint and the site description provided to the tools. If two constraints are of the same priority and cover the same path, then the last constraint in the constraint file will override any other constraints that overlap (unlike in XACT 5.x/6.x, where the "tightest" specification would be used).

Within Constraint Sources

Within constraint sources, the following priorities apply.

- TIG (Timing Ignore)—the highest priority
- FROM *source* THRU *point* TO *destination* specification

  The priority of each type of FROM THRU TO specification is as follows (highest priority is listed first).

  FROM USER1 THRU USER_T TO USER2 specification (USER1 and USER2 are user-defined groups)

FROM USER1 THRU USER_T TO FFS specification
or
FROM FFS THRU USER_T TO USER2 specification
(FFS is any pre-defined group)

FROM FFS THRU USER_T TO FFS specification

- FROM *source* TO *destination* specification

  The priority of each type of FROM TO specification is as follows (highest priority is listed first).

  FROM USER1 TO USER2 specification

  FROM USER1 TO FFS specification
  or
  FROM FFS TO USER2 specification

  FROM FFS TO FFS specification

- PERIOD specification
- "Allpaths" type constraints—the lowest priority

Layout constraints also have an inherent precedence which is based on the type of constraint and the site description provided to the tools. If two constraints are of the same priority and cover the same path, then the last constraint in the constraint file will override any other constraints that overlap.

Layout Constraints

The mapping constraints in the following example illustrate some of the capabilities for controlling the implementation process for a design. The OPTIMIZE attribute is attached to the block of logic associated with the instance GLUE. All the combinatorial logic in the GLUE block is optimized for speed (minimizing levels of logic) while other aspects of the design are processed by the default mapping algorithms (assuming the design based optimization switches are not issued).

```
# Mapping Constraint

  INST GLUE OPTIMIZE = SPEED ;
```

```
# Layout Constraint
#   LOC a pin

  NET IOBLOCK/DATA0_IN LOC = P12 ;

#   Reserve a pin (Pin 24 should be unused)

  CONFIG PROHIBIT = P24 ;
```

The layout constraint in the previous example illustrates the use of a full hierarchical path name for the net named DATA0_IN in the application of the I/O location constraint. In this example, IOBLOCK is a hierarchical boundary which contains the net DATA0_IN. Location constraints applied to "Pad nets" are used to constrain the location of the PAD itself, in this case to site P12.

Note: If the design contains a PAD, the constraint could have been just as easily applied to it directly (some design flows do not provide explicit I/O pads in the design netlist).

Efficient Use of Timespecs and Layout Constraints

The previous section described how to constrain your design's timing. The following sections summarize the available constraints.

Xilinx recommends defining design requirements at the highest level of abstraction first, and then fine tuning the timing requirements using more specific TIMESPECs as necessary. This methodology is recommended for the following reasons.

- The use of explicit Timegroups causes slower runtimes than the use of implicit timegroups arising from the use of constraints such as PERIOD.
- The processing of larger Timegroups takes longer than the processing of smaller Timegroups.
- The use of many specific Timespecs results in slower runtimes than the use of a smaller set of more general Timespecs.

Standard Block Delay Symbols

The "Timing Symbols and Default Values" table lists the block delay symbols and their description. There is a one-to-many correspondence between these symbol names and data book symbol names. For those symbols listed as having a default value of disabled, no timing analysis will be performed on paths which have segments

composed of symbol path. As an example, paths which have a set/reset to output path will not be analyzed. Any of the block delays (Symbol) listed in the table may be explicitly enabled or disabled using the PCF.

The following gives an example of the PCF syntax which would be used to enable the path tracing for all paths which contain RAM data to out paths. Note that this PCF directive is placed in the user section of the PCF.

```
SCHEMATIC END;

// This is a PCF Comment line
// Enable RAM data to out path tracing

ENABLE = ram_d_o;
```

Table H-1 Timing Symbols and Default Values

Symbol	Default	Description
reg_sr_q	Disabled	Set/Reset to output propagation delay.
lat_d_q	Disabled	Data to output transparent latch delay.
ram_d_o	Disabled	RAM data to output propagation delay.
ram_we_o	Enabled	Ram write enable to output propagation delay.
tbuf_t_o	Enabled	TBUF tri-state to output propagation delay.
tbuf_i_o	Enabled	TBUF input to output propagation delay.
io_pad_I	Enabled	IO pad to input propagation delay.
io_t_pad	Enabled	IO tri-state to pad propagation delay.
io_o_I	Enabled	IO output to input propagation delay. Disabled for tri-stated IOBs.
io_o_pad	Enabled	IO output to pad propagation delay.

Table of Supported Constraints

For further explanation and examples of each of the constraints, see the *Libraries Guide*.

Table H-2 Constraint Applicability

	Origin of Constraint	
Constraint	**Schematic**	**UCF***
BASE	Y	N
BLKNM	Y	Y
BUFG	Y	Y
CLKDV_DIVIDE	Y	Y
COLLAPSE	Y	Y
COMPGRP	N	N
CONFIG***	Y	N
DECODE	Y	Y
DIVIDE1_BY DIVIDE2_BY	Y	N
DOUBLE	Y	N
DRIVE	Y	Y
DROP_SPEC	N	Y
DUTY_CYCLE_CORRECTION	Y	Y
EQUATE_F EQUATE_G	Y	N
FAST	Y	Y
FILE	Y	N
FREQUENCY	N	N
HBLKNM	Y	Y
HU_SET	Y	Y
INIT	Y	N
INIT_0x	Y	N
INREG	Y	Y
IOB	Y	Y

Table H-2 Constraint Applicability

	Origin of Constraint	
Constraint	**Schematic**	**UCF***
KEEP	Y	Y
LOC	Y	Y
LOCATE	N	N
LOCK	N	N
MAP	Y	Y
MAXDELAY	Y	Y
MAXSKEW	Y	Y
MEDDELAY	Y	Y
NODELAY	Y	Y
NOREDUCE	Y	Y
OFFSET	N	Y
OPTIMIZE	Y	Y
OPT_EFFORT	Y	Y
OUTREG	Y	Y
PATH	N	N
PART	Y	Y
PENALIZE TILDE	N	N
PERIOD	Y	Y
PIN	N	N
PRIORITIZE	N	N
PROHIBIT	Y	Y
PWR_MODE	Y	Y
RLOC	Y	Y
RLOC_ORIGIN	Y	Y
RLOC_RANGE	Y	Y
S(ave) Net Flag Attributes	Y	Y
SITEGRP	N	N
SLOW	Y	Y

Table H-2 Constraint Applicability

	Origin of Constraint	
Constraint	**Schematic**	**UCF***
STARTUP_WAIT	Y	Y
TEMPERATURE	Y	Y
TIG	Y	Y
Timegroup Attributes	Y	Y
TNM	Y	Y
TNM_NET	Y	Y
TPSYNC	Y	Y
TPTHRU	Y	Y
TSidentifier	Y	Y
U_SET	Y	Y
USE_RLOC	Y	Y
VOLTAGE	Y	Y
WIREAND	Y	Y
XBLKNUM	Y	Y

*The UCF and NCF files use the same constraints with the exception of the INIT constraint, which can only be used in NCF files.

** Use cautiously—while constraint is available, there are differences between the UCF/NCF and PCF syntax.

*** The CONFIG attribute configures internal options of an XC3000 CLB or IOB. Do not confuse this attribute with the CONFIG primitive, which is a table containing PROHIBIT and PART attributes.

UCF Syntax Examples

The following sections summarize the functions of timespecs.

PERIOD Timespec

The PERIOD spec covers all timing paths that start or end at a register, latch, or synchronous RAM which are clocked by the reference net (excluding pad destinations). Also covered is the setup

requirement of the synchronous element relative to other elements (ex. flip flops, pads, etc...).

Note: The default unit for time is nanoseconds.

NET clk20MHz PERIOD = 50 ;
NET clk50mhz TNM = clk50mhz ;
TIMESPEC TS01 = PERIOD clk50mhz 20 ;

FROM TO Timespecs

FROM TO style timespecs can be used to constrain paths between time groups.

Note: The RAMS, FFS, PADS, and LATCHES keywords are predefined time groups used to specify all elements of each type in a design.

TIMESPEC TS02 = FROM PADS TO FFS 36 ;
TIMESPEC TS03 = FROM FFS TO PADS 36 ns ;
TIMESPEC TS04 = FROM PADS TO PADS 66 ;
TIMESPEC TS05 = FROM PADS TO RAMS 36 ;
TIMESPEC TS06 = FROM RAMS TO PADS 35.5 ;

OFFSET Timespec

To automatically include clock buffer / routing delay in your "PADS TO *synchronous element* or *synchronous element* TO PADS timing specifications, use OFFSET constraints instead of FROM TO constraints.

- For an input where the maximum clock-to-out (Tco) of the driving device is 10 ns.

  ```
  NET in_net_name OFFSET=IN:10:AFTER:clk_net ;
  ```

- For an output where the minimum setup time (Tsu) of the device being driven is 5 ns.

  ```
  NET out_net_name OFFSET=OUT:5:BEFORE:clk_net ;
  ```

Timing Ignore

If you can ignore timing of paths, use Timing Ignore (TIG).

Note: The "*" character is a wild-card that can be used for bus names. A "?" character can be used to wild-card one character.

- Ignore timing of net *reset_n*

  NET reset_n TIG ;

- Ignore *data_reg(7:0)* net in instance *mux_mem*

  NET mux_mem/data_reg* TIG ;

- Ignore *data_reg(7:0)* net in instance *mux_mem* as related to a TIMESPEC named TS01 only

  NET : mux_mem/data_reg* : TIG = TS01 ;

- Ignore *data1_sig* and *data2_sig* nets

  NET : data?_sig : TIG ;

Path Exceptions

If you have multi-cycle FF to FF paths, you can create a time group using either the TIMEGRP or TNM statements.

Warning: Many VHDL/verilog synthesizers do not predictably name flip flop Q output nets. Most synthesizers do assign predictable instance names to flip flops, however.

Example 1: TIMEGRP

TIMEGRP slowffs = FFS(inst_path/ff_q_output_net1* inst_path/ff_q_output_net2*);

Example 2: TNM Attached to Instance

INST inst_path/ff_instance_name1_reg* TNM = slowffs ;
INST inst_path/ff_instance_name2_reg* TNM = slowffs ;

Example 3:

If a FF clock-enable is used on all flip flops of a multi-cycle path, you can attach TNM to the clock enable net.

Note: TNM attached to a net "forward traces" to any FF, LATCH, RAM, or PAD attached to the net.

NET ff_clock_enable_net TNM = slowffs ;

Example 4:

Example of using "slowffs" timegroup, in a FROM:TO timespec, with either of the three timegroup methods previously shown.

TIMESPEC TS10 = FROM slowffs TO FFS 100 ;

Miscellaneous Examples

- Assign an IO pin number or place a basic element (BEL) in a specific CLB. BEL = FF, LUT, RAM, and so on.

  INST io_buf_instance_name LOC = P110 ;
  NET io_net_name LOC = P111 ;
  INST instance_path/BEL_inst_name LOC = CLB_R17C36 ;

- Prohibit IO pin C26 or CLB_R5C3 from being used.

  CONFIG PROHIBIT = C26 ;
  CONFIG PROHIBIT = CLB_R5C3 ;

- Assign an OBUF to be FAST or SLOW.

  INST obuf_instance_name FAST ;
  INST obuf_instance_name SLOW ;

- Constrain the skew or delay associate with a net.

  NET any_net_name MAXSKEW = 7 ;
  NET any_net_name MAXDELAY = 20 ns;

- Declare an IOB input FF delay (default = MAXDELAY).

Note: MEDDELAY/NODELAY can be attached to a CLB FF that is pushed into an IOB by the "map -pr i" option.

  INST input_ff_instance_name MEDDELAY ;
  INST input_ff_instance_name NODELAY ;

- Constraint priority in your .ucf file is as follows.

  Highest

  1. Timing Ignore (TIG)
  2. FROM : THRU : TO specs
  3. FROM : TO specs lowest
  4. PERIOD specs

Refer to the *Libraries Guide* for additional timespec features.

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 I

Glossary

This appendix contains definitions and explanations for terms used in the Quick Start Guide for Xilinx Alliance Series.

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.

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.

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 EPIC.

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).

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.

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.

Index

H

I

K

L

P

R

S

T

U

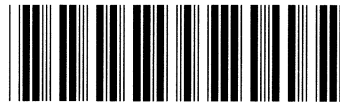
V

W

X

The Programmable Logic Company℠

2100 Logic Drive
San Jose, CA 95124-3400
Tel: 408-559-7778
Fax: 408-559-7114
www.xilinx.com



0401756