

Quick Start Guide for Xilinx Alliance Series vM1.2

vM1.2 Software Installation

Core Technology Tutorial

How This Release Works

Cadence Interface Notes

***FPGA Express Interface
Notes***

***Mentor Graphics Interface
Notes***

Synopsys Interface Notes

Viewlogic Interface Notes

LogiBLOX

Instantiated Components

Quick Start Guide



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

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



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

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

IBM is a registered trademark and PC/AT, PC/XT, PS/2 and Micro Channel are trademarks of International Business Machines Corporation. DASH, Data I/O and FutureNet are registered trademarks and ABEL, ABEL-HDL and ABEL-PLA are trademarks of Data I/O Corporation. SimuCad and Silos are registered trademarks and P-Silos and P/C-Silos are trademarks of SimuCad Corporation. Microsoft is a registered trademark and MS-DOS is a trademark of Microsoft Corporation. Centronics is a registered trademark of Centronics Data Computer Corporation. PALASM is a registered trademark of Advanced Micro Devices, Inc. UNIX is a trademark of AT&T Technologies, Inc. CUPL, PROLINK, and MAKEPRG are trademarks of Logical Devices, Inc. Apollo and AEGIS are registered trademarks of Hewlett-Packard Corporation. Mentor and IDEA are registered trademarks and NETED, Design Architect, System Architect, QuickSim, QuickSim II, and EXPAND are trademarks of Mentor Graphics, Inc. Sun is a registered trademark of Sun Microsystems, Inc. SCHEMA II+ and SCHEMA III are trademarks of Omation Corporation. OrCAD is a registered trademark of OrCAD Systems Corporation. Viewlogic, Viewsim, and Viewdraw are registered trademarks of Viewlogic Systems, Inc. CASE Technology is a trademark of CASE Technology, a division of the Teradyne Electronic Design Automation Group. DECstation is a trademark of Digital Equipment Corporation. Synopsys is a registered trademark of Synopsys, Inc. Verilog is a registered trademark of Cadence Design Systems, Inc.

Xilinx, Inc. does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx, Inc. reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx, Inc. will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx, Inc. devices and products are protected under one or more of the following U.S. Patents: 4,642,487; 4,695,740; 4,706,216; 4,713,557; 4,746,822; 4,750,155; 4,758,985; 4,820,937; 4,821,233; 4,835,418; 4,853,626; 4,855,619; 4,855,669; 4,902,910; 4,940,909; 4,967,107; 5,012,135; 5,023,606; 5,028,821; 5,047,710; 5,068,603; 5,140,193; 5,148,390; 5,155,432; 5,166,858; 5,224,056; 5,243,238; 5,245,277; 5,267,187; 5,291,079; 5,295,090; 5,302,866; 5,319,252; 5,319,254; 5,321,704; 5,329,174; 5,329,181; 5,331,220; 5,331,226; 5,332,929; 5,337,255; 5,343,406; 5,349,248; 5,349,249; 5,349,250; 5,349,691; 5,357,153; 5,360,747; 5,361,229; 5,362,999; 5,365,125; 5,367,207; 5,386,154; 5,394,104; 5,399,924; 5,399,925; 5,410,189; 5,410,194; 5,414,377; 5,422,833; 5,426,378; 5,426,379; 5,430,687; 5,432,719; 5,448,181; 5,448,493;

5,450,021; 5,450,022; 5,453,706; 5,466,117; 5,469,003; 5,475,253; 5,477,414; 5,481,206; 5,483,478; 5,486,707; 5,486,776; 5,488,316; 5,489,858; 5,489,866; 5,491,353; 5,495,196; 5,498,979; 5,498,989; 5,499,192; 5,500,608; 5,500,609; 5,502,000; 5,502,440; RE 34,363, RE 34,444, and RE 34,808. Other U.S. and foreign patents pending. Xilinx, Inc. does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. Xilinx, Inc. assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx, Inc. will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

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

Copyright 1991-1996 Xilinx, Inc. All Rights Reserved.

Preface

This quick start guide does what its title purports, provides an overview of the features and additions to Xilinx's newest product: vM1.2 Software. The primary focus of this guide is the Core Technology tools used to implement a design.

Chapter 1, “vM1.2 Software Installation,” introduces the new and enhanced features of the M1 software, and gives instructions on the installation of this latest Xilinx software on workstations, and PCs.

Chapter 2, “Core Technology Tutorial,” provides a tutorial, which in the terms of our engineer-writer, exercises all of the features of the M1 design flow.

Chapter 3, “How This Release Works,” looks at the in-depth capability, and flexibility of the Xilinx software.

There are seven appendices, five of them devoted each to an interface, one to LogiBLOX (including HDL), and finally one to frequently instantiated components.

Conventions

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.

`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 choose one or more.

`lowpwr = {on|off}`

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

`symbol editor_name [bus|pins]`

- 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;`

Contents

Chapter 1 **vM1.2 Software Installation**

Introduction	1-1
M1 Software Supported Families	1-1
Supported Netlists	1-2
M1 Software Instruction Volumes	1-2
New and Enhanced Tools	1-2
Design Manager	1-3
EPIC - Physical Design Editor	1-3
Flow Engine	1-3
Guide for Incremental Design Changes	1-4
Hardware Debugger	1-4
LogiBLOX	1-4
Multi-Pass PAR	1-5
PROM File Formatter	1-5
Re-Entrant Routing	1-5
Timing Analyzer	1-5
Timing Specification Performance	1-6
Third Party Synthesis	1-6
Installation	1-6
Obtaining and Setting Up Licenses	1-7
Setting Up Your License File	1-7
M1 Requirements for Workstations	1-8
M1 Installation on Workstations	1- 9
Installing the Core Technology Software	1-10
Installing the CAE Interface and Libraries	1-10
Installing the On-line Documentation	1-10
Variable Settings for Workstations	1-10
Verifying DynaText Variable Settings - Workstation	1-11

Chapter 1 vM1.2 Software Installation continued

Verifying Core Technology	
Software Installation - Workstations	1-12
Setting Up LogiBLOX for Workstations	1-13
M1 Requirements for PCs	1-13
M1 Installation on PC	1-14
Installing CAE Interface, and Libraries	1-14
Installing Workview Office Toolset	1-14
Installing On-line Documentation	1-15
Variable Settings for PCs	1-15
Setting Up LogiBLOX for Use With Workview Office	1-16

Chapter 2 Core Technology Tutorial

Step 1: Invoking Design Manager,	
Creating an Implementation Project	2-2
Step 2: Creating Design Version, Implementation Revisions.....	2-3
Step 3: Mapping a Design.....	2-5
Step 4: Using Timing Analyzer to Evaluate	
Block Delays After Mapping.....	2-11
Step 5: How to Place and Route a Design.....	2-13
Step 6: Evaluating With Worst Case Timing	2-17
Step 7: Using the Flow Engine to Create	
Timing Simulation Data.....	2-19
Step 8: Using the Flow Engine to Create Configuration Data	2-21
Step 9: Using the PROM File Formatter to Create PROM Files ...	2-23

Chapter 3 How This Release Works

Starting Xilinx Tools	3-1
Creating a Project	3-4
Implementing a Design	3-4
Translate	3-4
Map	3-4
Place and Route.....	3-5
Configure	3-5
Analyzing Reports	3-6
Translation Report	3-6
Map Report	3-6
Place and Route Report.....	3-7
Pad Report.....	3-7

Chapter 3 How This Release Works continued

Selecting Options	3-8
Using Constraint Files	3-9
Design, Netlist, User, and Physical Constraints	3-9
Creating a User Constraint File	3-10
Guiding an Implementation	3-13
Exact Guide Mode	3-14
Leveraged Guide Mode	3-14
Static Timing Analysis	3-14
Static Timing Analysis After Map	3-15
Static Timing Analysis After Place and Route	3-15
Summary Timing Reports	3-16
Detailed Timing Analysis	3-16
Creating Simulation Files	3-17
When Can Simulation Data Be Created	3-17
Creating Timing Simulation Data	3-17
Creating Functional Simulation Data	3-18
Downloading a Design	3-19
Creating a PROM	3-19
In-Circuit Debugging	3-19
Advanced Implementation Flows	3-19
Re-Entrant Route	3-20
Multi-Pass Place and Route	3-21

Appendices

Cadence Concept and Verilog Interface Notes	A-1
FPGA Express Interface Notes	B-1
Mentor Graphics Interface Notes	C-1
Synopsys Interface Notes	D-1
Viewlogic Interface Notes	E-1
LogiBLOX	F-1
Instantiated Components	G-1

List of Figures

2-1	Project "count8-tutorial" Begun	2-3
2-2	Design Manager Status Bar	2-5
2-3	Toolbox shown in horizontal placement	2-5
2-4	Flow Engine With "count8.ucf" Selected	2-7
2-5	Control Panel	2-8
2-6	Flow Engine Shows Completed Map Stage	2-9
2-7	Report Browser	2-10
2-8	Implementation Options	2-14
2-9	Place and Route Stage Completed	2-15
2-10	Reports Available After Place & Route Stage	2-16
2-11	Completed Timing Stage	2-20
2-12	Completed Configure Stage	2-22
2-13	PROM Properties Dialog With Single PROM	2-24
2-14	Split PROM Dialog With Multiple PROMs	2-25
3-1	M1 Software Design Flow	3-2
3-2	Detailed Design Flow	3-3
3-3	Design Manager Menu	3-4
3-4	Flow Engine indicates completion of each design segment	3-5
3-5	Report Browser	3-6
3-6	Options Dialog	3-8
A-1	Cadence/Verilog Interface and M1 Design Flow	A-4
A-2	Top Portion of Figure A-1	A-5
A-3	Bottom Portion of Figure A-1	A-6
B-1	FPGA Express/M1 Design Flow	B-5
C-1	Mentor/M1 Software Flow	C-4
D-1	Synopsys/M1 Software Design Flow	D-4
E-1	Viewlogic/M1 Core Technology Design Flow	E-5

List of Tables

1-1	Supported Netlists.....	1-2
1-2	M1 Memory Requirements (Workstations).....	1-8
1-3	M1 Memory Requirements for Windows 95 and Windows NT.....	1-13
G-1	Startup Library Component.....	G-1
G-2	BSCAN Library Components	G-2
G-3	Readback Library Components.....	G-3
G-4	RAM and ROM Library Components	G-4
G-5	Global Buffers Library Components	G-5
G-6	Fast Output Primitives.....	G-6
G-7	IOB Components.....	G-8

Chapter 1

vM1.2 Software Installation

Version M1.2 software is hereinafter referred to as M1 Software.

Introduction

Welcome to Xilinx's newest software release, vM1.2, known as M1 Software. The following enhancements and additions, made possible through a new core design, provide development designers with an improved suite of tools for implementing Programmable Logic Devices (PLDs).

Note: Complete compatibility with previous XACT releases XNF and LCA logical and physical design files has been preserved. The M1 Software may be used with the Xilinx Foundation Package tools.

The step-by-step installation procedure, entitled Installation, begins on page 1-6.

M1 Software Supported Families

M1 Software supports the following families with earlier version compatibility: 3000A, 3100A, 4000E, 4000L, 5000L, 5200, 7300, and 9000. Two new families are also supported by the M1 Software: 4000EX and 4000XL. For further details about earlier version compatibility, refer to the appropriate CDROM (supplied with your Xilinx software). Xilinx manuals are divided by subject matter, and more than several volumes can apply to one or more products. Technical information for all but the latest products (XC4000EX and XC4000XL high density FPGA families) is included in the Xilinx "Programmable Logic Data Book".

Supported Netlists

Refer to the following table for netlists supported by the M1 Software.

Table 1-1 Supported Netlists

Netlists	Variations
EDIF	Multiple variations are accepted, including: <ul style="list-style-type: none">• SEDIF• EDIF• EDN
XNF	Multiple variations are accepted, including: <ul style="list-style-type: none">• SXNF• XFF• XNF• XTF

Note: All designs must be created using the Xilinx Unified Libraries, located in the “Libraries Guide”.

M1 Software Instruction Volumes

For a more detailed listing of documentation on this software release, please refer to the on-line documentation shipped via enclosed CDROMs, or visit our Web site at <http://www.xilinx.com>

New and Enhanced Tools

New and enhanced design features included in the M1 Software are listed in the order in which their descriptions appear in this section.

- Design Manager
- EPIC - the Physical Design Editor
- Flow Engine
- Guide for Incremental Design Changes
- Hardware Debugger
- LogiBLOX
- Multi-Pass PAR

- Netlist Support
- PROM File Formatter
- Re-Entrant Routing
- Timing Analyzer
- Timing Specification Performance

Design Manager

The Xilinx Design Manager is the graphical interface which manages the data file versions implemented during the design process. Results of these implementations are made available in reports and may be accessed through the Design Manager's Report Browser.

Design Manager also provides on-screen pushbutton access to the other Xilinx tools, such as the Flow Engine, Timing Analyzer, PROM File Formatter, EPIC Report Browser, and various navigation and information tools.

EPIC - Physical Design Editor

Editor for Programmable Integrated Circuits (EPIC), a graphical editor, provides a view of the physical implementation of your Xilinx design. EPIC is new, and in function, a replacement for XDE/EDITLCA. Focus features of EPIC include:

- Viewability of CLB mapping, placement and routing
- Timing analysis with critical paths highlighted
- Modification capability of the placement and routing of your Xilinx design
- Ability to create custom macros to be incorporated in your Xilinx design

Flow Engine

The Flow Engine displays and then executes all the steps needed to implement a Xilinx design. The Flow Engine:

- translates the design netlist
- maps the logic to CLBs

- places and routes the design
- creates a configuration file which downloads a design to a part.
- creates static timing reports and timing simulation netlists in the following formats: VHDL (Vital), Verilog, EDIF, or XNF

Guide for Incremental Design Changes

Xilinx's Guide for Incremental Design Changes has been enhanced in the M1 Software to support guided mapping, and guided place and route functions.

Added to the guide in the M1 software is the Leverage Guide mode. This mode was specifically designed to accommodate the design iteration that requires more than minor changes, or the instance where a module has changed entirely due to the synthesis of your design.

Hardware Debugger

The Hardware Debugger is an addition to the existing software. The Hardware Debugger downloads a configuration file to a single FPGA, or to a daisy chain of FPGAs through the Xchecker, Serial, or Parallel download cables. When used with the Xchecker cable, the Hardware Debugger can read back the state (logic levels) of the design signals inside the FPGA, and thus, enable in-circuit design debugging.

LogiBLOX

New in the M1 Software is LogiBLOX. The successor to X-BLOX, LogiBLOX can be used to create the following types of modules, among others:

- ROMs
- RAMs
- counters
- comparators
- decoders
- modules for synthesis
- modules for schematic capture designs

- modules of behavioral simulation for fast functional simulation

LogiBLOX can be used as a stand-alone for synthesis designs, or can be integrated with Viewlogic or Mentor Graphics schematic capture software. Refer to Appendix F of this manual for further details.

Multi-Pass PAR

The place and route (PAR) software allows multiple place and route iterations to be run:

- on a single machine
- on a UNIX network
- on multiple machines in parallel

The multi-pass feature achieves optimum performance and efficiency, utilizing maximum CPU time to achieve design results more readily and quickly.

PROM File Formatter

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.

Re-Entrant Routing

Once a place and route result is found that is close to meeting the desired specifications, notably those of timing, or is close to being completely routed, the implementation process can be re-entered to continue the routing process. This allows option and specification modifications during the routing stage of design implementation.

In addition to facilitating design changes, re-entrant routing significantly reduces CPU time of re-compiles.

Timing Analyzer

The Timing Analyzer produces a report on:

- overall design performance
- timing specifications performance

- specific path performance

Timing analysis can be performed on the block delays of a just-mapped design, or on the block and route delays of a design already placed and routed.

Timing Specification Performance

Xilinx M1 Software now supports timing-driven placement and routing. The timing specification capability has been enhanced to incorporate greater precision. Features include

- new timing constraints
- implicit and explicit overrides of conflicting constraints
- override or overlap capability of constraints by slower or faster constraints that are either 1) narrower or 2) have a higher priority

Third Party Synthesis

Third party synthesis or schematic capture tools create netlists. Once a netlist format is selected, the working directory is set automatically to the resident netlist directory.

For information on the following Xilinx supplied interface tools, refer to:

Cadence	Appendix A
FPGA Express	Appendix B
Mentor Graphics	Appendix C
Synopsys	Appendix D
Viewlogic	Appendix E
LogiBLOX	Appendix F
Instantiated Components	Appendix G

Installation

Optimum use and operation of your M1 design tools is best guaranteed through correct installation of the M1 Software on recommended hardware, with recommended operating memory capacity. If you experience problems with either the installation, operation, or verification of your installation, please contact the Xilinx Technical Support hotline.

North America:

Telephone: 1-800-255-7778

Fax: 1-408-879-4442

The following presents recommended hardware and memory, installation instructions for workstations and PCs, and verification of that installation. Information is provided first for workstations; then, for PCs. The following licensing information applies to both workstation and PC users.

Obtaining and Setting Up Licenses

Before running your M1 Software, you will need to obtain a license from Xilinx. The license and authorization codes are in the *license.dat* file you will receive from your Xilinx Customer Service representative. Contact Xilinx Customer Service at the above number, Monday through Friday, from 8:00 am. to 5:00 pm, Pacific Standard Time.

When you telephone/fax for your license, please be prepared to provide the following information:

- product name(s)
- serial number from your product registration card

Workstation Users:

- Hostname and hostid of the server where the license(s) file will be installed

PC Users:

- Volume Serial Number of the C:\ drive of the PC the M1 license server will be running on. To find this number, open an MS-DOS session and, from the C:\ drive, type VOL.

Please give the C:\ drive Volume Serial Number even if the Xilinx tools are to be installed to a different drive letter.

Setting Up Your License File

Once you receive your license and authorization codes from Customer Service:

Workstation Users: You can add license and authorization codes to an existing *license.dat* file and restart the license server. To start a

new license server, you can also create a new *license.dat* file in the "data" sub-directory of the Xilinx software tree.

PC Users: Place your license.dat file in the C:\xilinx\data\license.dat directory. If you decide to place this file in a different location, you must also modify the licence file by entering the path and file, e.g., c:\xilinx\license.dat. The Xilinx license.dat file does not authorize the Workview Office software. Current Workview Office users will use their existing Workview Office 7.2 license.dat to authorize the Workview Office 7.3 tools. Refer to the following paragraphs, "Variable Settings for PCs", for more information about using multiple license.dat files.

For more information on how to create or modify a license.dat file and start or restart a license server, see the "Installation" section of the Release Notes.

M1 Requirements for Workstations

The M1 Software supports the following workstation architectures and operating systems:

- SunOS 4.1.3, and 4.1.4
- Solaris 5.4 and 5.5
- Ultra Sparc (or equivalent)
- HP-UX HP 10.1
- HP715 (or equivalent)

Table 1-2 M1 Memory Requirements

Xilinx Device	RAM	Swap Space
4000EX (refer to note below) XC3000A/L XC3100A/L XC4000E/L XC4028EX through XC4036EX XC4005XL through XC4028XL XC5200/L XC7300 XC9500/F (small devices only)	64 MB	64 MB

Table 1-2 M1 Memory Requirements

Xilinx Device	RAM	Swap Space
XC4036XL through XC4062XL XC9500/F (large devices only)	128 MB	128 MB

Note: The values given in the above table are for typical designs, and include the loading of the operating system. Additional memory may be required for certain “boundary-case” or “pathological” designs, as well as for concurrent operation of other applications (e.g., MS Word or Excel).

Xilinx recommends that 4000EX designs be compiled using an Ultra Sparc, HP715, or equivalent machine type. 64MB of RAM as well as 64MB of swap space is required to compile 4000EX designs, but Xilinx recommends that 128MB of RAM, plus corresponding swap space, be used.

M1 Installation on Workstations

Refer to the following table, and also to the installation section of the “Release Notes”.

All workstation installations must be done while logged in with ‘root’ authority.

Required Installation, Verification	
For SunOS 4.1.3 and 4.1.4, Solaris 5.4 and 5.5, Ultra Sparc (or equivalent), HP-UX, HP 10.1, and HP715 (or equivalent)	
<ul style="list-style-type: none"> • Setup license file • Core Technology Package software • CAE interface and Libraries • On-line Documentation 	<ul style="list-style-type: none"> • Obtain license and authorization code for each product • Set variable settings • Verify DynaText and variable settings • Verify Core Technology

Installing the Core Technology Software

Installation of “Core Technology” software is completed in two steps:

1. Mount the CDROM labeled “Core Technology.”
2. Run “install” located in the CDROM root directory.

Note: For Solaris machines, a separate ‘install’ program is located in a subdirectory called ‘cdrom0’.

For detailed information on how to mount and unmount a CDROM, and how to run the various installation programs, see the installation section of the “Release Notes”.

Installing the CAE Interface and Libraries

Installation of library viewing interface software is completed in two steps:

1. Mount the CDROM labeled, “CAE Interfaces”.
2. Run “install” located in the CDROM root directory.

Installing the On-line Documentation

Installation of “On-line Documentation” on workstations is completed in two steps:

1. Mount the CDROM labeled “On-line Documentation”.
2. Run “install” located in the CDROM root directory.

Variable Settings for Workstations

The M1 Software requires environment variables to be set in order to run properly.

The Core Technology software, running on either a Sun or HP workstation requires the following variables be set:

- XILINX
- path
- m1_license

- LM_LICENSE_FILE
- LD_LIBRARY_PATH (SunOS and Solaris only)
- SHLIB_PATH (HP-UX only)

These variables should be set in the following manner:

```
setenv XILINX <installation_path_of_Xilinx_tools>
set path = ($XILINX/bin/platform_name $path)
setenv ml_license $XILINX/data/license.dat
setenv LM_LICENSE_FILE $ml_license
```

For SunOS and Solaris only:

```
setenv LD_LIBRARY_PATH $XILINX/bin/platform_name
```

For HP-UX only:

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

For example:

```
setenv XILINX /usr/xilinx
set path = ($XILINX/bin/sun $path)
setenv ml_license $XILINX/data/license.dat
setenv LM_LICENSE_FILE $ml_license
setenv LD_LIBRARY_PATH $XILINX/bin/sun
```

Verifying DynaText Variable Settings - Workstation

The M1 Software is located in the \$XILINX/data/dtext directory on the CDROM. The documentation must be viewed using the DynaText browser supplied on the CDROM and installed during the Core Technologies software installation. The browser is installed in the \$XILINX/bin/platform_name directory. The DynaText environment is defined by the .ebtrc file. There is a .ebtrc file for each environment supported by the M1 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. This variable should be set in the following manner:

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

For example:

```
setenv EBTRC $XILINX/bin/sol/ebtrc
```

To launch the DynaText browser issue the following command:

dttext

If the EBTRC or XILINX environment variables are not set correctly, DynaText will return the following errors:

```
DynaText (TM) v2.3, Oct 1 1994, Copyright(c)1990-1994
Issue #639
Electronic Book Technologies, Inc.
All Rights Reserved.
Error : Can't find config file '.ebtrc'.
Error : Your .ebtrc does not point to a 2.x DATA_DIR
```

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

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 “Installation” section of the Release Notes.

Verifying Core Technology Software Installation - Workstations

Once you have set up all the required environment variables, it is a good idea to verify that they have been set correctly.

If your setup is correct, PAR will run normally and return the command line information.

If a variable is incorrectly set, refer to the following examples to help you debug your setup.

In each of the examples, the test command is:

par

Example 1: If your setup has a variable incorrectly set, you will get an error like the following:

```
par: Command not found
```

In this case, you would need to check your ‘path’ and ‘XILINX’ environment variables.

Example 2: Another error PAR may return is:

```
ld.so: libbasgi.so.1: not found
```

In this case, you would need to check the LD_LIBRARY_PATH environment variable if running SunOS or Solaris, or the SHLIB_PATH environment variable if running HP-UX.

Example 3: A fatal error PAR may return is:

```
FATAL ERROR: The XILINX environment variable must be
set. Exiting...
```

In this case, you need to check the XILINX environment variable

Setting Up LogiBLOX for Workstations

Refer to Appendix F of this manual, and the section entitled “Setting Up LogiBLOX on Workstations”.

M1 Requirements for PCs

The M1 Software supports the following PC operating systems:
Windows 95 (3.0 or later) and Windows NT (4.0 or later)

Table 1-3 M1 Memory Requirements for Windows 95 & Windows NT

Xilinx Device	RAM	Virtual Memory
XC3000A/L XC3100A/L XC4003E/L through XC4008E/L XC4005XL through XC4008XL XC5200/L XC7300 XC9500/F (small devices only)	32 MB	32 MB
XC4010E/L through XC4025E/L XC4028EX through XC4036EX XC4010XL through XC4028XL XC5206/L - XC5215/L XC9500/F (medium devices only)	64 MB	64 MB
XC4036XL through XC4062XL XC9500/F (large devices only)	128 MB	128 MB

Note: The values given in the above table are for typical designs, and include the loading of the operating system. Additional memory may be required for certain “boundary-case” or “pathological” designs, as well as for concurrent operation of other applications (e.g., MS Word or Excel).

M1 Installation on PC

Refer to the following table and also to the installation section in the “Release Notes” for further details.

Required Installation, Verification	
For Windows 95 (3.0 or later), and Windows NT (4.0 or later)	
<ul style="list-style-type: none">• Setup license file• Core Technology Package software, CAE Interface, and Libraries.• Workview Office 7.3 (only if using Viewlogic)• On-line Documentation	<ul style="list-style-type: none">• Obtain license and authorization code for each product• Set variable settings• Set up LogiBLOX for use with ViewDraw

Installing CAE Interface, and Libraries

Installation of “Core Technology” software is completed in two steps:

1. Insert the CDROM labeled “Core Technology” in the CDROM drive.
2. Run *setup.exe* located in the CDROM root directory.

Installing Workview Office Toolset

Installation of the Workview Office Toolset is completed in two steps:

1. Insert the CDROM labeled “Workview Office 7.3” in the CDROM drive.
2. Run *wvoinst.exe* located in the CDROM root directory.

Installing On-line Documentation

Installation of “On-line Documentation” is completed on the PC in two steps:

1. Insert the CDROM labeled “On-line Documentation” in the CDROM drive.
2. Run *setup.exe* located in the CDROM root directory.

Variable Settings for PCs

The M1 Software requires environment variables to be set in order to run properly.

The Core Technology software requires the following variables be set:

- XILINX
- PATH
- LM_LICENSE_FILE

These environment variables should be set in the following manner:

```
set XILINX=c:\xilinx
set PATH=c:\xilinx\bin\nt;%PATH%
set LM_LICENSE_FILE=c:\xilinx\data\license.dat
```

If Workview Office has also been installed, the following variables must be set:

- XILINX
- PATH
- LM_LICENSE_FILE
- WDIR
- VANTAGE_VSS (ViewSynthesis only)
- VANTAGE_CC (ViewSynthesis only)

These environment variables should be set in the following manner:

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

```
set LM_LICENSE_FILE=c:\wvoffice\standard  
  \license.dat,;c:\xilinx\data\license.dat  
set WDIR=c:\wvoffice\standard  
set VANTAGE_VSS=c:\wvoffice\v  
set VANTAGE_CC=c:\wvoffice\msvcnt\bin\cl
```

Note: The **LM_LICENSE_FILE** variable uses a comma followed by a semicolon (;) to separate the two paths.

For Windows NT 4.0 users only:

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

For Windows 95 users only:

Run SYSEDIT to open the AUTOEXEC.BAT file, and verify the environment settings are as shown above.

The variables listed assume that the Xilinx M1 tools have been installed in the C:\XILINX directory and the Workview Office tools have been installed in the C:\WVOFFICE directory. If these default paths have been changed, the environment settings must change accordingly.

Setting Up LogiBLOX for Use With Workview Office

One final setup procedure is needed if you plan to use the LogiBLOX Graphical User Interface with ViewDraw, the Workview Office schematic entry tool. For a complete description of this setup, please see “Setting up LogiBLOX on PCs” in Appendix F.

Chapter 2

Core Technology Tutorial

This tutorial demonstrates the M1 Core Technology implementation flow. The design, a simple 8-bit counter with asynchronous clear and clock enable, was compiled using Synopsys FPGA Compiler and is described by a Synopsys Xilinx netlist file.

In an effort to exercise the entire flow, the tutorial passes timing specifications from Synopsys to the M1 Core Technology tools using a netlist constraints file and incorporates user constraints using a user constraints file. Also included is a physical constraints file which the tutorial uses to evaluate the timing of the design beyond the specifications supplied by the netlist constraints file.

The files should be copied from the following directory located on the M1 Core Technology CDROM:

`/xbbs/tutorial/qstart/XILINX/orig`

The files you should copy to an empty working directory are:

count8.sxnf	Synopsys Xilinx Netlist file
count8.ncf	Netlist Constraints File
count8.ucf	User Constraints File
count8.pcf	Physical Constraints File

Step 1: Invoking Design Manager, Creating an Implementation Project

The M1 Core Technology 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 M1 implementation tools used to complete a design. To begin this tutorial, change directories to the area containing your copy of the design files and invoke the Design Manager in the background.

1. On a workstation enter the following at the command line prompt:

```
dsgnmgr &
```

On a PC, invoke the Design Manager by double-clicking on its icon.

When running the Design Manager for the first time, there are no projects available. To create an implementation project for the tutorial design, proceed to 2 below.

2. Select File->New Project

The New Project dialog is displayed 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. The comment field is where you may enter a brief notation about the design being processed.

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

The Browse dialog is displayed with the default file type set to EDIF Files. The design of this tutorial was created by Synopsys FPGA Compiler. Therefore, open an SXNF file.

4. Click on the List Files of Type pulldown list box and select XNF Files (*.xnf, *.xtf, *.sxnf) to change the file filter to the desired file type.
5. Select the count8.sxnf file and click on OK to accept the input netlist.

The Browse dialog is closed and the New Project dialog is updated to include the specified input netlist. By default, the Work Directory is set to the directory containing the input design. This can be set to another directory if so desired. Being that we have created a new directory and copied the files, we can use the same directory to hold the implementation project and resulting output files.

6. Place the cursor in the Comment field and enter
- tutorial
7. Click OK. This closes the New Project dialog and updates the Design Manager with the specified project. Refer to Figure 2-1.

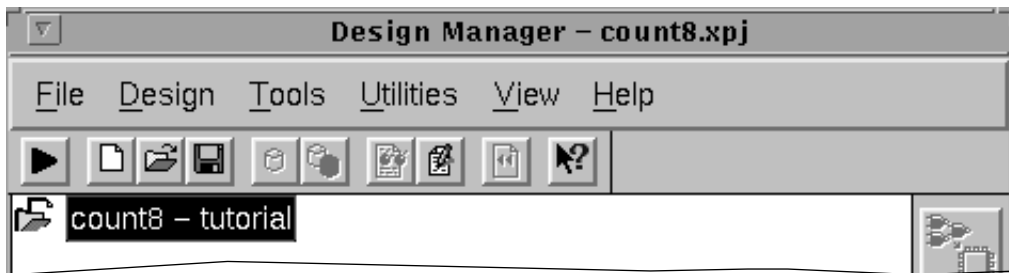


Figure 2-1 Project "count8 - tutorial" Begun

Note: None of the icons in the Toolbox on the right side of the Design Manager are active. To use these tools, a design version and an implementation revision must be created.

Step 2: Creating Design Versions, Implementation Revisions

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. Remember that the Design Manager only keeps track of the Xilinx created files; the input design is not archived with the imple-

mentation data stored in the Xilinx implementation project area. Because you might try different implementation strategies, you will also have several revisions for a single version.

Note: This tutorial covers the basics. For detailed information on flows and implementation methodologies using the M1 Core Technology tools, see the Development System Reference Guide.

1. Select Design->New Version to create the first design version.

The New Version dialog is displayed with the default version being 'ver1'.

2. Click OK to create the new version.

The Design Manager now shows 'ver1' under the count8 design project.

3. Select Design->New Revision to create the first implementation revision.

The New Revision dialog is displayed with the default revision being 'rev1'. Notice that the Part field already contains the value 'XC4028EX-3-PG299'. This information was found by the Design Manager in the input netlist when the version was created.

4. Click OK to create the new revision.

The Design Manager now shows 'rev1' under the initial version of the count8 project. (No figure provided, refer to actual program.) The status of the revision is noted in parenthesis as (New, OK). The first portion (i.e., new) refers to the state of the design and will be updated throughout the tutorial as we complete the different compilation stages. The second portion (i.e., OK) is the status of the current state. Currently the design is new and no errors or warnings have been generated.

At the bottom of the Design Manager is the status bar (refer to Figure 2-2). The status bar contains information such as the current project, target device, and currently selected version ->revision pair. The leftmost portion of the status bar (not shown in the figure) is used by the tool to give information on what is currently selected by the cursor.

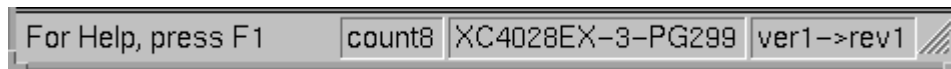


Figure 2-2 Design Manager Status Bar

The Toolbox, located on the right side of the Design Manager becomes active with your first implementation revision. Refer to Figure 2-3. The icons contained in the Toolbox are only active when a version->revision pair is selected. The design is now ready to be implemented. Icons in the Toolbox represent, left to right, Flow Engine, Timing Analyzer, PROM File Formatter, Hardware Debugger, and EPIC Design Editor.

Note: The Toolbar has drag-and-drop capability.



Figure 2-3 Toolbox shown in horizontal placement

Step 3: Mapping a Design

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

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

The Flow Engine is invoked using the default flow settings. Stages or processes in a design are given graphical representation in the upper half of the Flow Engine screen. The status, when complete, of each stage is also depicted.

2. The first stage is to translate the design from the input netlist format to an internal format. Then, with all the input netlists

translated and merged, the basic logic elements of the design are mapped into CLBs and IOBs. Once the resources of the target device have been allocated, the design can be evaluated, and then placed and routed. The placed and routed design is then verified and configuration (configure) data can be created to download to the target system.

The Flow Engine gives you, the designer, complete control on how the design is processed. Typically you will set all desired implementation options and run through the entire flow. However, to demonstrate the flexibility of the Flow Engine, and best provide you with a working knowledge, this tutorial proceeds through each stage of the flow. For reference, refer to the flow diagram shown in Chapter 3 of this manual.

3. Select Setup->Options to open the implementation Options dialog.

With the Options dialog you specify any guide files, user constraints files, optional processing targets. You can also access the implementation and configuration templates.

4. Click on the Browse button to the right of the User Constraints field.

The Browse dialog is displayed with the working directory selected in the Directories field.

5. Select the count8.ucf file and click OK to accept the user constraints.

The User Constraints field of the Options dialog is updated with the specified constraints file. At this point we want to map the design's logic and, therefore, do not need to modify the templates or specify any additional targets.

6. Click OK to close the Options dialog.

Refer to Figure 2-4. The Options dialog is closed and the status bar at the bottom of the Flow Engine is updated with the specified user constraints file.

In this tutorial we want to map the design only. Therefore, specify that the Flow Engine stop after the map process by setting a break point.

7. Click on the stop sign toolbar icon (refer to Figure 2-4).

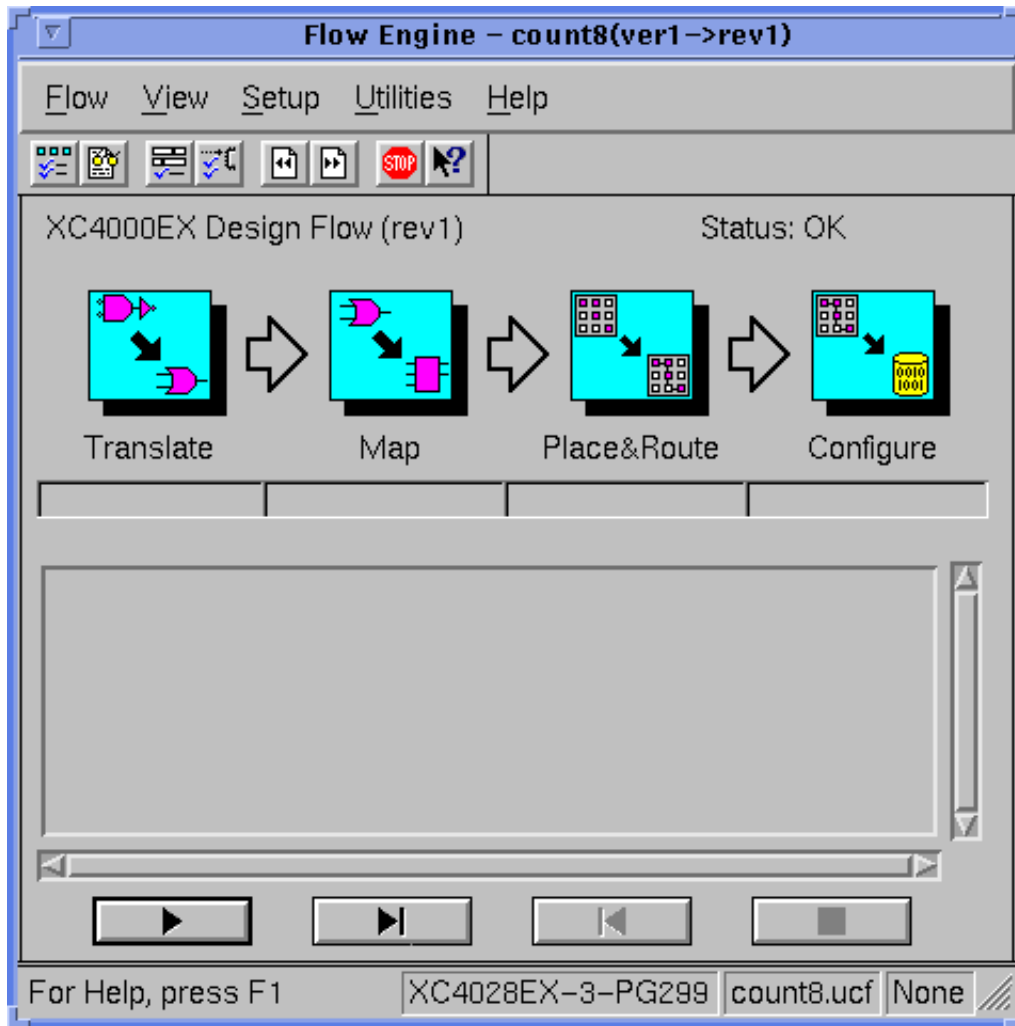


Figure 2-4 Flow Engine With “count8.ucf” Selected

When you select STOP, the Stop After dialog is displayed with the default setting of Configure. The list box will only contain possible break points from the current state of the design.

Because we have yet to process the design, all possible break points are available.

8. Select Map in the list box and click on Ok to accept the break point.

Note, on your screen, the stop sign is now added to the flow between the Map and Place & Route phases. This “stop” allow us to run the flow until the break point. There are several ways to begin the implementation process. The Flow->Run and Flow->Step commands can be used, or their equivalent control buttons, shown in Figure 2-5, can be selected.



Figure 2-5 Control Panel, left to right, Play, Step Forward, Step Backward, and Stop (Press <F1> for on-line Help).

9. Click on the 'play' control button (far left on the Control Panel, Figure 2-5) to start the process.

The Flow Engine first runs ngdbuild.

- Ngdbuild converts all the input design netlists and then writes the results into a single merged file.
- Ngdbuild adds the user constraints file and any constraints found in any netlist constraints files to the merged netlist. This merged netlist fully describes not only the logic in the design but also any location and timing constraints.

Note: If you want to perform a functional simulation of the design, you can set a break point after the Translate phase 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 NGD2 program on the file to create the desired functional simulation data. For more information on the NGD2 programs, see the Development System Reference Guide.

Map is the next program run by the Flow Engine (refer to Figure 2-6).

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

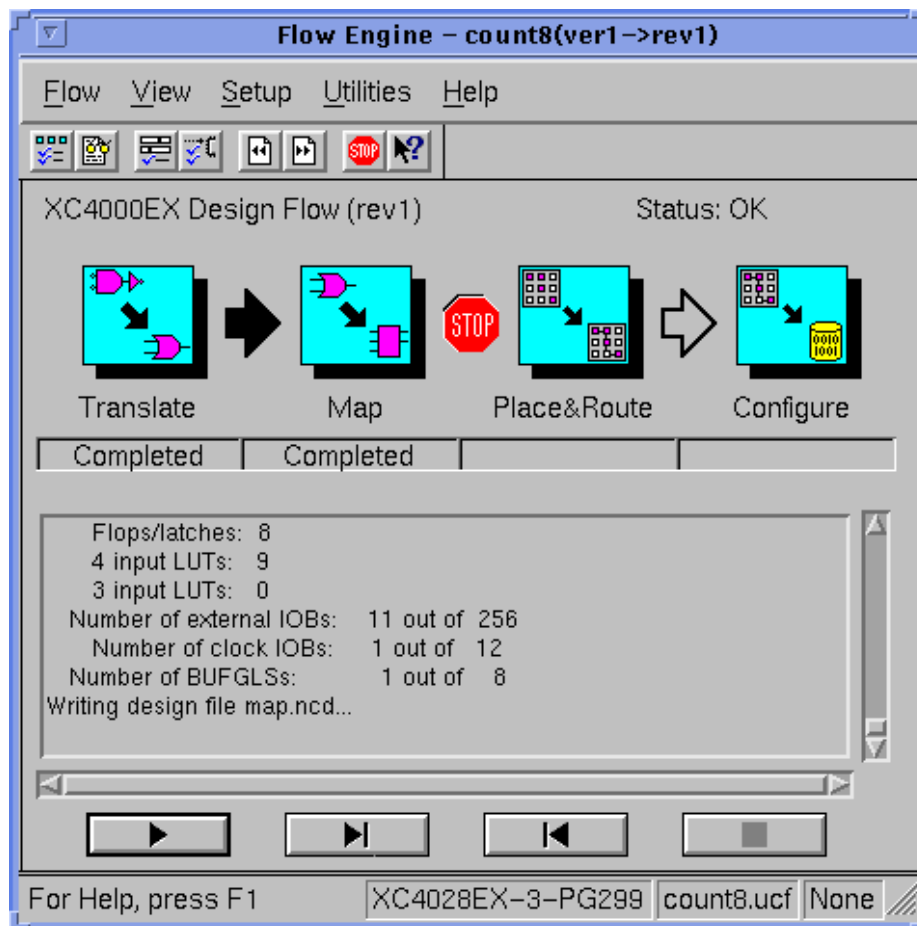


Figure 2-6 Flow Engine Shows Completed Map Stage

Once the Map phase is complete, we can review the currently available reports using the Report Browser.

10. Select Utilities->Report Browser to access the currently available reports.

The Report Browser is invoked containing the Translation and Map Report files. Reports that are new are denoted with a gold star in the upper left corner of the file icon.

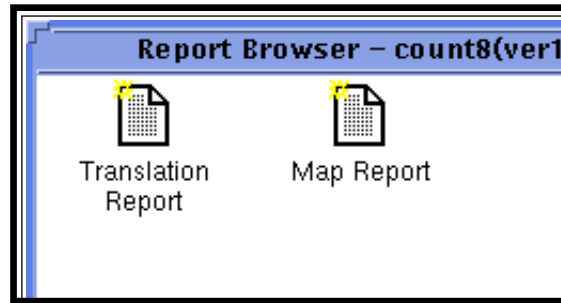


Figure 2-7 Report Browser shows reports available after map stage.

11. Double-click on the Map Report to review the Map process output.
 - **Map Report** contains information on how the target device resources were allocated, where user locked logic was placed, references to trimmed logic, and device utilization. For detailed information on the Map report, refer to the Development System Reference Guide.
 - **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.

Notice that the current version->revision now has the status of (Mapped, OK).

12. Close the map report. Then select Flow->Close to close the Flow Engine and the Report Browser.

The design (this tutorial design) has been mapped to the target architecture. We can now evaluate the various paths to ensure that they do not contain too many block delays.

Step 4: Using Timing Analyzer to Evaluate Block Delays After Mapping

After the design has been mapped, the Timing Analyzer can be used to evaluate the logical paths in the design. Because the design has not yet been placed and routed, no net delay information is known. *This means that the timing reports created will be based only on the logical block delays; all nets are estimated or reported as 0ns.*

Evaluating a design using only block delays gives you a preliminary look at how realistic your timing goals are, given the design's current logical implementation. A rough guideline (known as the 50/50 rule) is 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 10ns path after mapping should meet a 20ns timing constraint after it has been placed and routed.

1. To invoke the Timing Analyzer, click on the Timing Analyzer icon in the Toolbox.

The Timing Analyzer, once invoked, automatically loads the mapped netlist. Should the Timing Analyzer display a message about the default physical constraints file, simply click OK to dismiss it.

- The Timing Analyzer can be used to time paths by creating groups within the various dialogs and windows.
- You can also read in a physical constraints file containing timing groups and associated timing specifications to be used to time the paths in the design.

2. Select File->Open Physical Constraints.

The Open Physical Constraints dialog is displayed with the current version->revision selected in the Directories list box.

3. Change the directory selection in the Directories list box by double-clicking on this tutorial's working directory.

The timing.pcf physical constraints file supplied with the tutorial files should now appear in the File Name list box.

4. Select the *timing.pcf* file and click OK to accept the physical constraints file.

The status bar at the bottom of the Timing Analyzer will be updated with the specified physical constraints file.

5. Select Analyze->Timing Constraints to generate a report of the paths covered by the timing constraints specified in the *timing.pcf* physical constraints file.

The Timing Analysis In Progress dialog is displayed while the report is being generated. Should you desire to cancel the generation of a report, simply click on the Abort button.

The report generated is called an error report. Should any of the paths analyzed fail their corresponding timing requirement, a timing error will be generated. The default number of timing error(s) reported is 1. Because the timing requirements are not overly aggressive for the count8 design, no timing errors are generated.

Note: If a timing error is encountered when running a mapped timing analysis, the timing constraint is viewed as impossible to meet. This is because the mapped report only contains block delays and there must be some time allocated for actual net routing. Should you ignore this type of error, the place and route tools will not allow you to proceed with your implementation until the specification is relaxed or the number of block delay is reduced.

To get a more detailed report on the worst case path we can run a detailed (the following) report.

6. Select Analyze->All Paths to generate a detailed report on the longest path in the design.

The resulting report, see below, contains the following worst case path:

Path n172 to n166 contains 5 levels of logic:

Path starting from Comp: CLB.K (from n117)

To	Delay type	Delay(ns)	Physical Resource	Logical Resource(s)

CLB.XQ	Tcko	1.830R	n172	n172
CLB.F1	net	e 1.145R	n172	
CLB.COUT	Topcy	3.900R	n172	dd_25/plus/plus/u8/S0_1/CO_2

```

CLB.CIN    net          e   1.166R          dd_25/plus/plus/u8/S0_1/CO_2
CLB.COUT   Tbyp        0.350R   n170        dd_25/plus/plus/u8/S0_1/CO_4
CLB.CIN    net          e   1.166R          dd_25/plus/plus/u8/S0_1/CO_4
CLB.COUT   Tbyp        0.350R   n168        dd_25/plus/plus/u8/S0_1/CO_6
CLB.CIN    net          e   1.166R          dd_25/plus/plus/u8/S0_1/CO_6
Tsum+Tick  1.910R   n166        dd_25/plus/plus/u8/S0_1/CO_7
___-return55<7>
n165

```

Total (64.2% logic, 35.8% route)

12.983ns (to n117)

Once a design is mapped, we can roughly determine the performance of the design by applying the 50/50 rule. Recall, the 50/50 rule says that about 50% of any path delay will be due to block delays and 50% will be due to routing delays.

If we apply the 50% logic (block delays) , 50% routing rule, the worst case path should be about 17ns after routing the design, given that the block levels currently contribute about 8.5ns. The net delays listed are actually estimates of what might happen once the design is placed and routed.

7. Select File->Exit to close the Timing Analyzer.

As the Timing Analyzer closes, you can choose to save the generated reports or simply discard the results.

Step 5: How to Place and Route a Design

After you have evaluated the mapped design and feel that the block delays are reasonable given the specified timing goals, you will use the Flow Engine now to place and route the design.

The Flow Engine can run the place and route algorithms in several different ways.

- You can run with the timing constraints specified in the netlist and user constraints files

- Or you can instruct the place and route tools to ignore them.

Note: It is recommended that timing be ignored on the initial pass of the design in order to give you a look at what is possible in the least amount of processing time.

1. Click on the Flow Engine icon in the Toolbox to invoke the Flow Engine.

The Flow Engine is invoked in the mapped state (the same state the Flow Engine had when we closed it at the end of Step 3). First we must specify the desired options to run the place and route stage.

2. Select Setup->Options to display the Options dialog.

All the options for the various implementation programs can be found in the Implementation and Configuration templates.

3. Click on the Edit Template button to the right of the Implementation template pulldown list box.

The XC4000 Implementation Options dialog is displayed. There are four tabs that can be selected, the default tab being the Optimize & Map tab (refer to Figure 2-8).

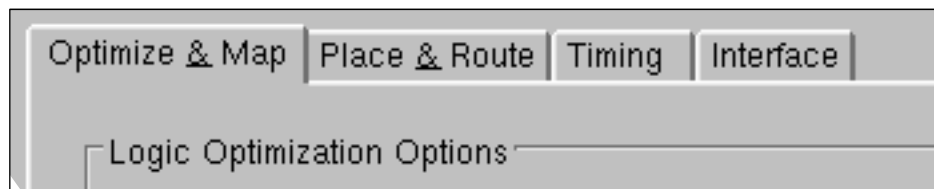


Figure 2-8 Implementations Options

4. Click on the Place & Route tab.

This tab allows you to control how hard the place and route tools work while implementing your design. The placer level, route passes, number of delay based clean-up passes, and whether or not to use timing constraints can be specified.

5. While holding the left mouse button on the placer slider, slide the control completely to the left.

If you wanted the tools to ignore the timing constraints found in the provided netlist constraints file, you could deselect the Use Timing Constraints During Place and Route check box (refer to your screen).

With the desired options set, proceed with the implementation flow.

6. Click OK to close the XC4000 Implementation Options dialog. Click OK to close the Options dialog.
7. Select Flow->Step to run only the Place & Route phase of the flow.

Even though we didn't specify a break point, the Flow Engine will stop after the Place & Route stage because we ran a single step. Refer to Figure 2-9.

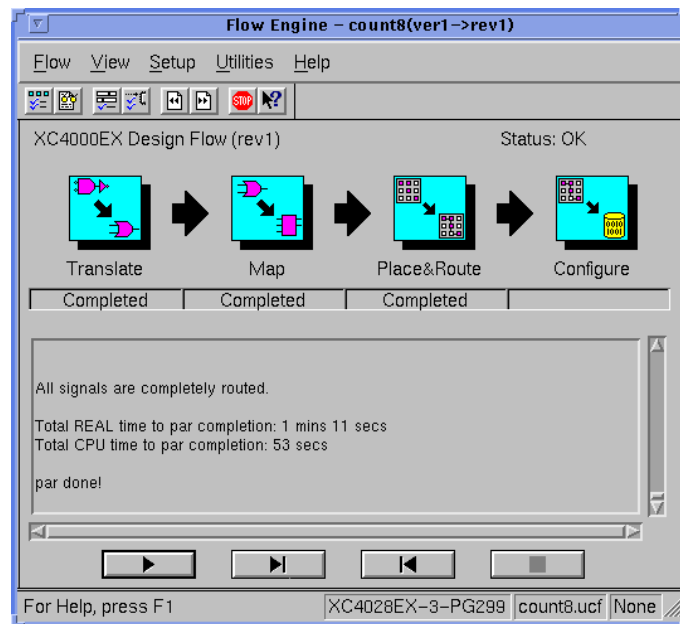


Figure 2-9 Place and Route Stage Completed

We will want to review the reports to make sure that the processing is being completed as expected. Status (located in the upper right of the Display Manager) can also be checked on to make sure that no errors were encountered.

8. Select Utilities->Report Browser to invoke the Report Browser.

Three additional reports are available for reviewing. Refer to Figure 2-10.

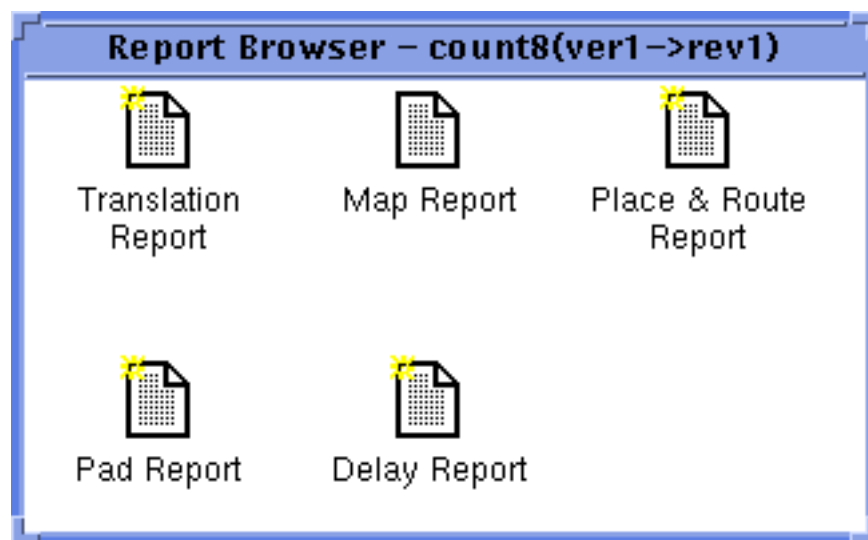


Figure 2-10 Reports Available After Place & Route Stage

The number of reports available for review has gone from two to five.

- **Delay Report** enumerates all the nets in the design and the delays of all the loads on the net. Use this report to help determine where you might have fanout or other design-dependent issues.
- **Map Report** (refer to Step 3 above for description).
- **Pad Report** contains a report of where each of the device pins were located in the device. Use this report to lock the pins of the design down as well as to verify that any pins that were locked down were placed in the correct place.

- **Place & Route Report** contains the information that was displayed in the Flow Engine log window. Use this report to make sure that the design successfully routed and that all the timing constraints were met.
 - **Translation Report** (refer to Step 3 above for description).
9. Select Flow->Close to close the Flow Engine and Report Browser.

Note: Status of the current version->revision now is: (Routed, OK).

Step 6: Evaluating With Worst Case Timing

With the design placed and routed, the Timing Analyzer can be used to evaluate the logical paths to ensure that the design meets the desired timing goals.

1. Click on the Timing Analyzer icon in the Toolbox. The Timing Analyzer is invoked.

The Timing Analyzer automatically loads the routed netlist. By default, the Timing Analyzer loads the current revision's physical constraints file which contains the constraints passed by Synopsys via the netlist constraints file.

2. Select Analyze->Timing Constraints.

The only constraint passed through the netlist constraints file is a period constraint on the design's global clock. This constraint specifies the following:

- all the flip-flop to flip-flop paths must be 50ns
- all pad to flip-flop and flip-flop to pad paths must be 25ns.

When the Timing Analyzer evaluates this type of constraint (a period constraint), it reports the worst path and notes any timing errors if any paths fail to meet the desired timing. Because the worst case path is a flip-flop to flip-flop path, we cannot discern the running time of the I/O paths.

To generate a more detailed report, we will use the *timing.pcf* file to evaluate the block levels of the mapped design.

3. Select File->Open Physical Constraints. The Open Physical Constraints dialog is displayed with the current version->revision selected in the Directories list box.

4. Change the directory selection in the Directories list box by double-clicking on the tutorial's working directory. The *timing.pcf* physical constraints file supplied with the tutorial files should now appear in the File Name list box.
5. Select the *timing.pcf* file and click OK to accept the physical constraints file.

Note: The status bar at the bottom of the Timing Analyzer is updated with the specified physical constraints file.

6. Select Analyze->Timing Constraints to generate a report of the paths covered by the timing constraints specified in the *timing.pcf* physical constraints file.

Using the *timing.pcf* physical constraints file not only tells us what the worst case flip-flop to flip-flop path delay is but also the worst case pad to flip-flop and flip-flop to pad path delays. For this implementation, the worst case pad to flip-flops path is 15.917ns and the worst case flip-flop to pad path is 7.630ns.

7. To get a more detailed report, select Analyze->All Paths. The resulting report contains the following worst case path:

Path n172 to n166 contains 5 levels of logic:

Path starting from Comp: CLB_R32C32.K (from n117)

To	Delay type	Delay(ns)	Physical Resource	Logical Resource(s)

CLB_R32C32.XQ	Tcko	1.830R	n172	n171
CLB_R32C32.G4	net	6.563R	n171	
CLB_R32C32.COUT	Topcy	3.900R	n172	dd_25/plus/plus/u8/S0_1/CO_2
CLB_R31C32.CIN	net	1.171R		dd_25/plus/plus/u8/S0_1/CO_2
CLB_R31C32.COUT	Tbyp	0.350R	n170	dd_25/plus/plus/u8/S0_1/CO_4
CLB_R30C32.CIN	net	1.171R		dd_25/plus/plus/u8/S0_1/CO_4
CLB_R30C32.COUT	Tbyp	0.350R	n168	dd_25/plus/plus/u8/S0_1/CO_6
CLB_R29C32.CIN	net	1.171R		dd_25/plus/plus/u8/S0_1/CO_6
	Tsum+Tick	1.910R	n166	dd_25/plus/plus/u8/S0_1/CO_7
___-return55<7>				

n165

Total (45.3% logic, 54.7% route) 18.416ns (to n117)

After mapping the design, the total delay with estimated net delays was 12.983ns. The block delays contributed about 8.5ns. Simply doubling the block delays would give 17ns, which is not far from what was achieved with the lowest place and route effort level for this simple design.

Note: As an exercise, you can create another revision and set the placer effort level to the maximum value. As a reference, running the design with the maximum placer effort and 1 delay based clean-up pass results in a worst case path time of 14.271ns. Given the block delays of 8.5ns, this is better than the 50% logic, 50% routing rule.

8. Select File->Exit to close the Timing Analyzer.

As the Timing Analyzer closes, you can choose to save the generated reports or simply discard the results.

Step 7: Using the Flow Engine to Create Timing Simulation Data

With the design placed and routed and the timing statically verified using the Timing Analyzer, you can create timing simulation data. Because the tutorial files were created using Synopsys, we will target VHDL as the backannotated format for the simulation data.

1. To invoke the Flow Engine, click on the Flow Engine icon in the Toolbox.
2. The Flow Engine is invoked in the routed state (the same state it was left in. To create the desired VHDL format simulation data, we must do two things.
 - a) First, select timing simulation data as a target.
 - b) Second, select VHDL as the output format for the data created. The first is accomplished in the Options dialog while the second option is set in the Implementation template.
3. Select Setup->Options to open the Options dialog.

The first option we need to specify in the Options dialog is the timing simulation data option.

4. Click on the Produce Timing Simulation Data checkbox to select it as a desired target.

With the target specified, we now need to specify the desired format of the data to be created.

5. Click on the Edit Template button to the right of the Implementation template pulldown list box. The XC4000 Implementation Options dialog is displayed. Because we are creating the data for a particular interface, the option we need to set is located in the Interface tab.
6. Click on the Interface tab at the top of the XC4000 Implementation Options dialog.
7. Select VHDL in the Format pulldown list box located in the Simulation Data Output section.
8. Click OK to close the XC4000 Implementation Options dialog. Click OK to close the Options dialog.

The flow is now updated to include the Timing phase. Refer to Figure 2-11.

Note: If you had created configuration data and then selected the Timing phase, the Flow Engine would have automatically backed up the process to after the Place & Route phase.

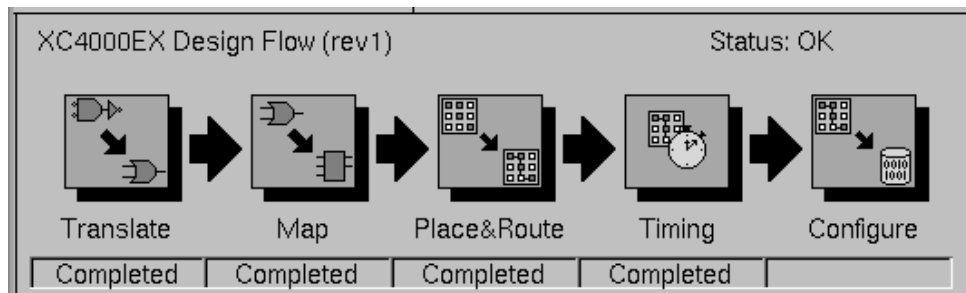


Figure 2-11 Completed Timing Stage

9. Select Flow->Step to create the desired timing simulation data.

Note: The Flow Engine runs ngdanno to create a backannotated NGD file. The NGD file can then be used as the input to any of the NGD2 programs to produce the desired simulation file format. For information on the NGD2 programs, see the Development System Reference Guide.

Because we specified VHDL, the Flow Engine runs NGD2VHDL and creates 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 for use in a third party simulation environment, these files are automatically copied to this tutorial's working directory.

Step 8: Using the Flow Engine to Create Configuration Data

After timing simulation has been performed and the static timing analysis numbers reviewed, download data can be created. First, a bitstream must be created for each device on the board. This is accomplished by running the Configure phase in the Flow Engine.

1. If you don't already have the Flow Engine running, invoke it on the Timed revision created in the previous step.

There are many configuration options available. As an introduction, we will set the input and output threshold levels to CMOS.

2. Select Setup->Options to open the Options dialog.
3. Click on the Edit Template button to the right of the Configuration template pulldown list box. The XC4000 Configuration Options dialog is displayed.
4. 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.
6. Click OK to close the Options dialog.

With the desired options set, we can now complete the implementation of this design revision by running the Configure phase.

7. Select Flow->Run to run the Configure phase. The Flow Engine runs a tool called bitgen to create the configuration data. Bitgen creates two files, count8.bit and count8.1l. The first file (count8.bit) is the actual configuration data while the second (count8.1l) is the logical allocation file used to determine where the probable points in the design are.

The *design.1l* file is used for performing device readback using the Hardware Debugger. For more information on device readback, see the Hardware Debugger User Guide.

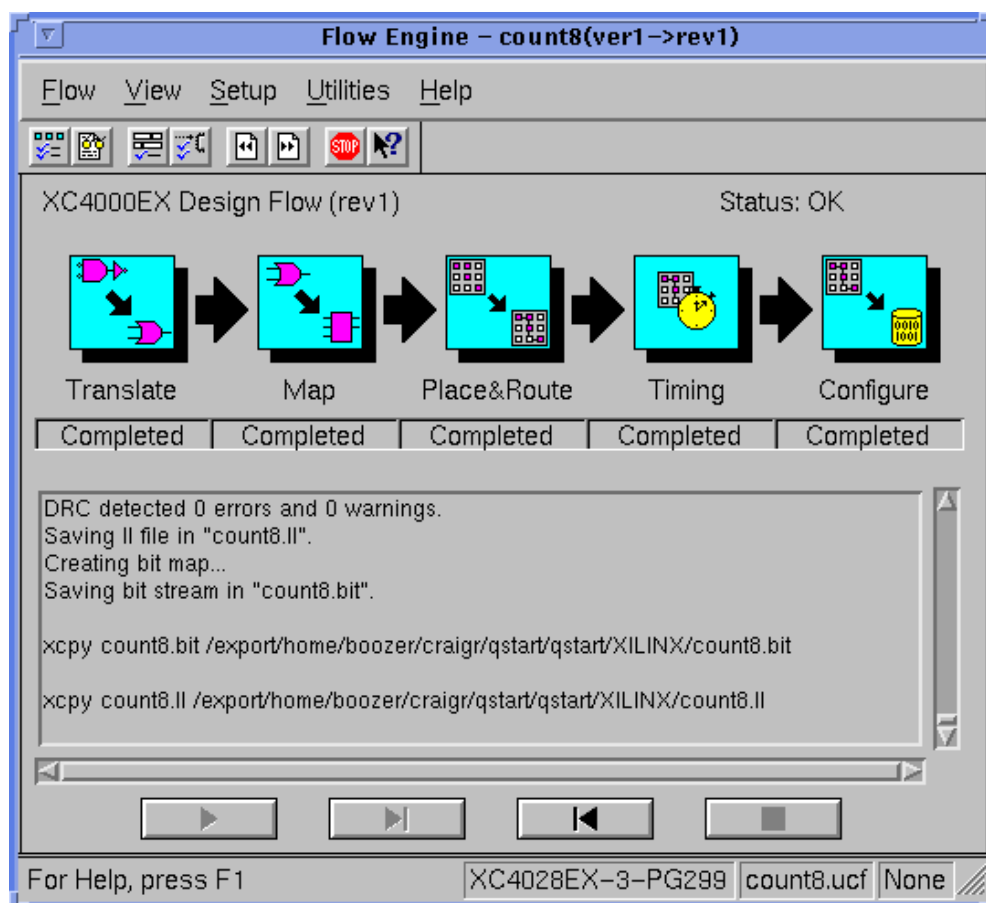


Figure 2-12 Completed Configure Stage

The Flow Engine saves the configuration options in the Bitgen Report. Review the report using the Report Browser and verify that the CMOS thresholds were in fact specified when creating the configuration data.

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

Step 9: Using the PROM File Formatter to Create PROM Files

If you are only going to be programming a single device using the Hardware Debugger, all that you need is the design.bit file. If you are going to be programming several devices in a daisy chain configuration, or will be programming your devices using a PROM, the PROM File Formatter must be used 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.

The Formatter is invoked with a default PROM already created containing the currently selected Configured version->revision. At this point you can either add additional bitstreams to the daisychain, create additional daisychains, remove the current bitstream and start from scratch, or immediately save the current PROM file configuration. Refer to Figure 2-13, PROM Properties Dialog.

The status bar at the bottom of the PROM File Formatter (PFF) denotes the PROM format, data format, current PROM size, and percentage of the selected PROM used by the current PROM configuration. 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 will show how to save the default PROM file.

Notice that the currently selected PROM is an XC17256. Because the target device for the tutorial was an XC4028EX, 668184 bits of data are needed in the PROM to hold the config-

uration bitstream. This means that the current device is 254% full. Obviously we will need to split the data across several PROMs. The PROM properties must be modified before the PROM can be saved.

2. Select File->PROM Properties to open the PROM Properties dialog.

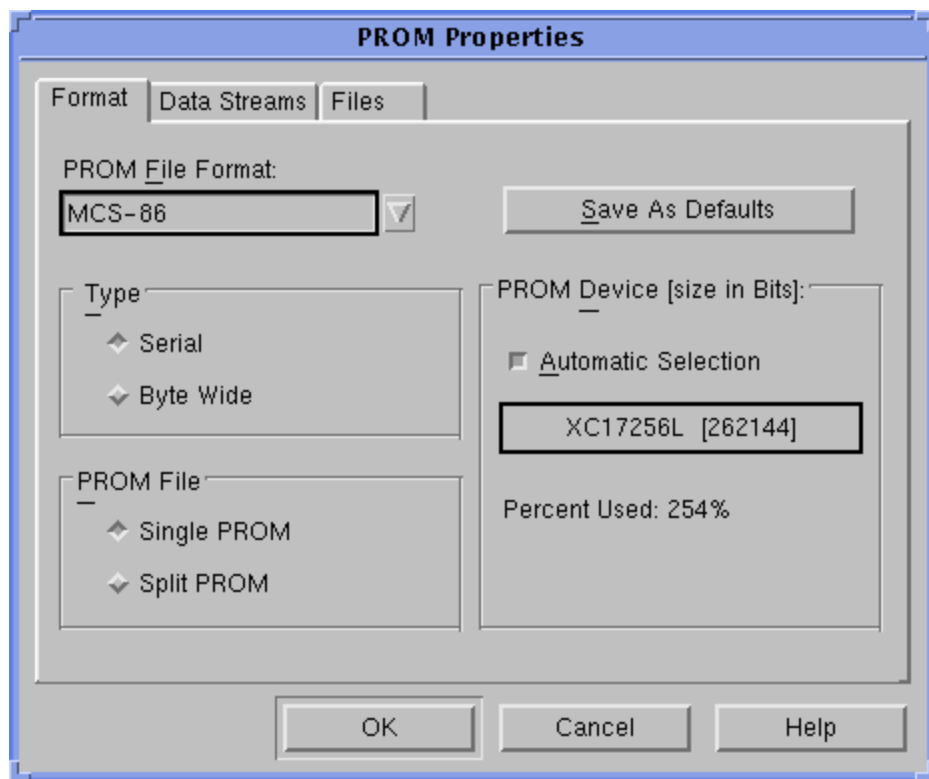


Figure 2-13 PROM Properties Dialog With Single PROM

The PROM Properties dialog can be used to select the PROM format as well as the PROM type used and the number of PROMS used to hold the desired data. Because we have more data than space available in the XC17256 we need to split the data into several individual PROMs.

3. Click on the Split PROM radio button.

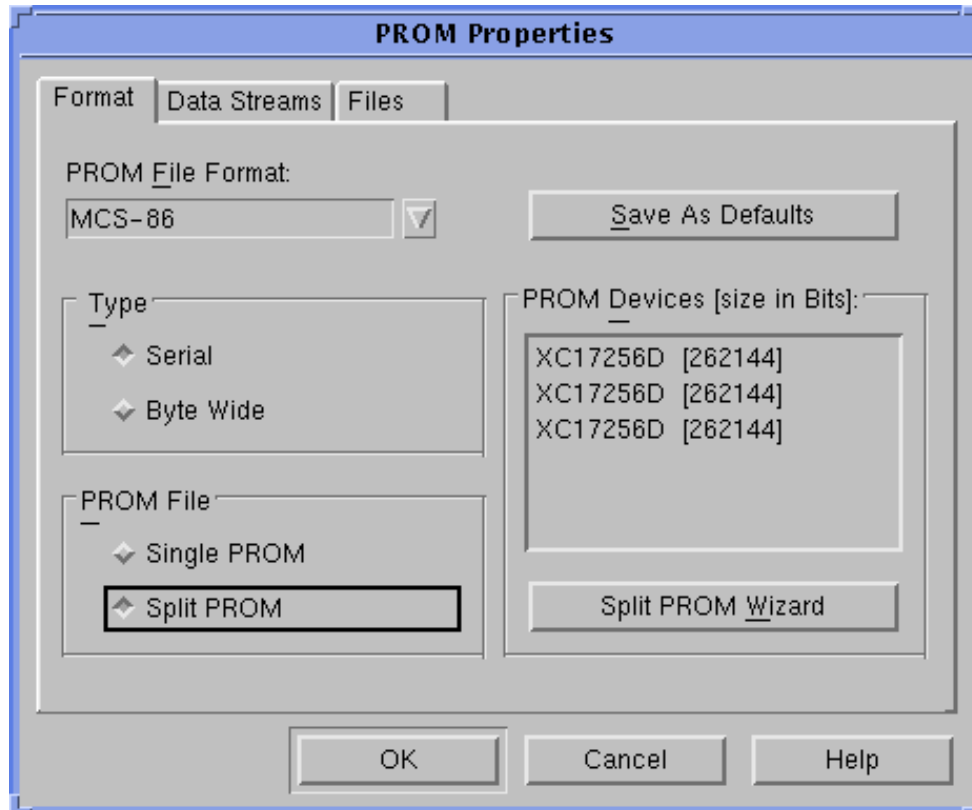


Figure 2-14 Split PROM Dialog With Multiple PROMs

Notice that the PROM Device area is modified and now shows multiple XC17256 devices. When the PROM configuration is saved, three individual files will be saved. The Split PROM Wizard button can be selected to invoke an automated wizard which will allow you to customize the types and quantities of PROMs used. For this tutorial, we will use the XC17256 PROMs already displayed.

4. Click OK to accept the PROM Properties.

5. Select File->Save to save the three PROM files.
6. Specify the working directory as the area where the PROM Description File will be saved.

The PROM File Formatter will save not only the PROM files, but also a PROM Description File. This PDF file can be opened if changes are needed.

7. Select File->Exit to close the PROM File Formatter.

This completes the tutorial.

Chapter 3

How This Release Works

This chapter explains how the M1 Software release of the Xilinx tools works. The standard flow from netlist to PROM file will be described, including discussions on options, reports, creating simulation netlists, constraints, and guided implementations. Advanced flows like re-entrant routing and multi-pass place and route will also be discussed.

This chapter contains the following sections:

- Standard Implementation Flow

- Advanced Implementation Flows

Refer during discussion to Figures 3-1 and 3-2. Figure 3-1 provides an overview of the M1 Software Design Flow; Figure 3-2, a detailed design flow.

Starting Xilinx Tools

To start the Xilinx tools double click on the Design Manager icon, or at a command line type:

```
dsgnmgr
```

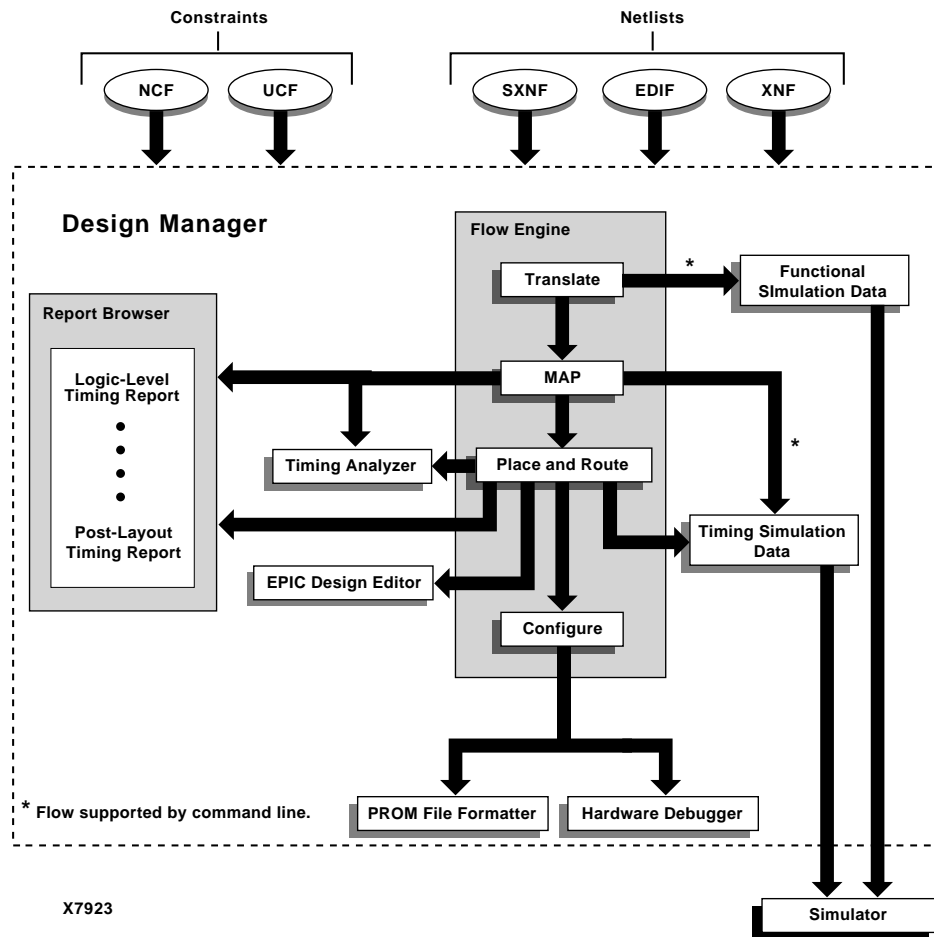
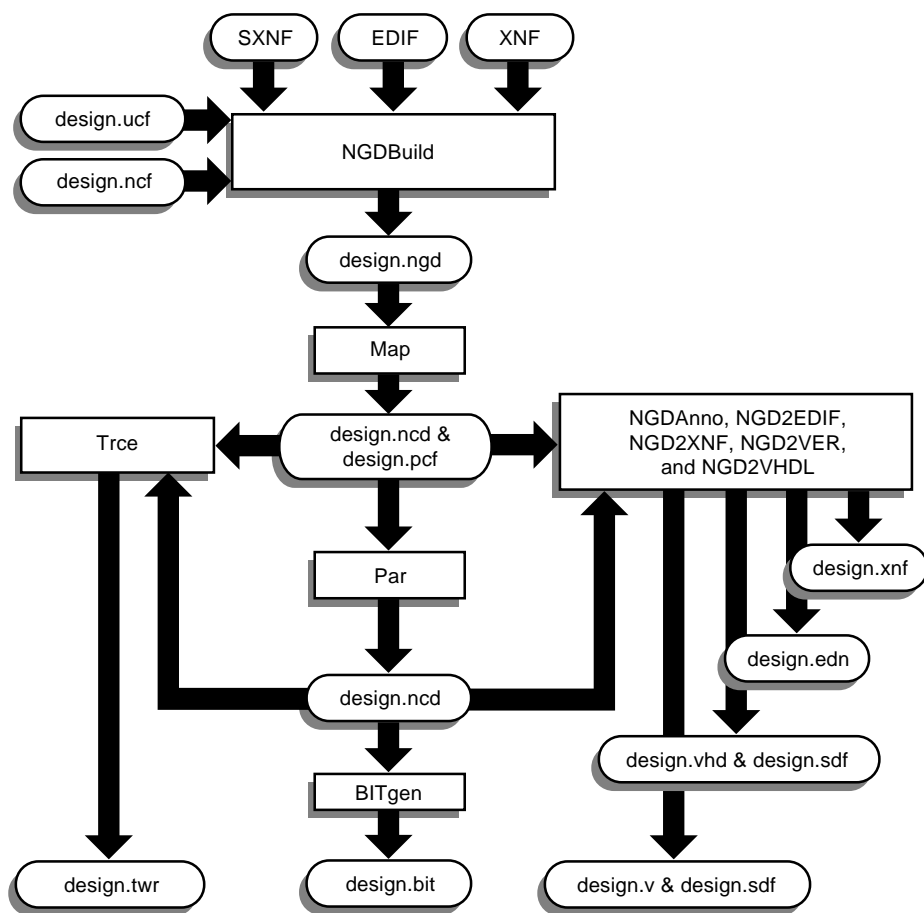


Figure 3-1 M1 Software Design Flow



X8037

Figure 3-2 Detailed Design Flow

Creating A Project

From Design Manager (refer to Figure 3-3), select Design->Implement. Click on the Browse button to select the top level input netlist. Third party synthesis and schematic capture tools create the netlists. Once the netlist has been selected, the working directory is automatically set to the directory in which the selected netlist resides.

For information on how to run the Xilinx supplied interface tools for Synopsys, Viewlogic, Mentor Graphics, or Cadence designs, see the appropriate appendix at the end of this book.

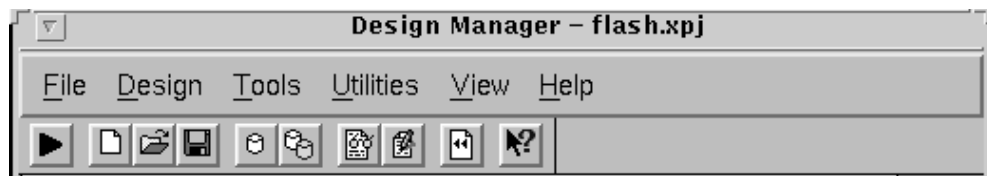


Figure 3-3 Design Manager Menu

Implementing A Design

From the Design Manager menu, select Design->Implement. In the Implement dialog 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 the design.

Translate

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

Map

The next step is Map. Map optimizes the gates and trims unused logic in the merged NGD netlist. Map also maps the designs logic resources and performs a physical design rule check. Logic in the design is mapped to resources on the silicon and a physical design

rule check is performed. The Map process is accomplished by running the MAP executable.

Place and Route

Once the design is mapped, the Flow Engine places and routes the design. In the place stage, all logic blocks, including the configurable Logic Blocks (CLB) and input/output blocks (IOB) structures, are assigned a specific location on the die. If there are timing constraints on particular logic components, the placer tries 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. If there are timing constraints on particular logic components, the router tries to minimize those delays by choosing a faster interconnect. The place and route (PAR) process is accomplished by running the PAR executable. Refer to Figure 3-4.

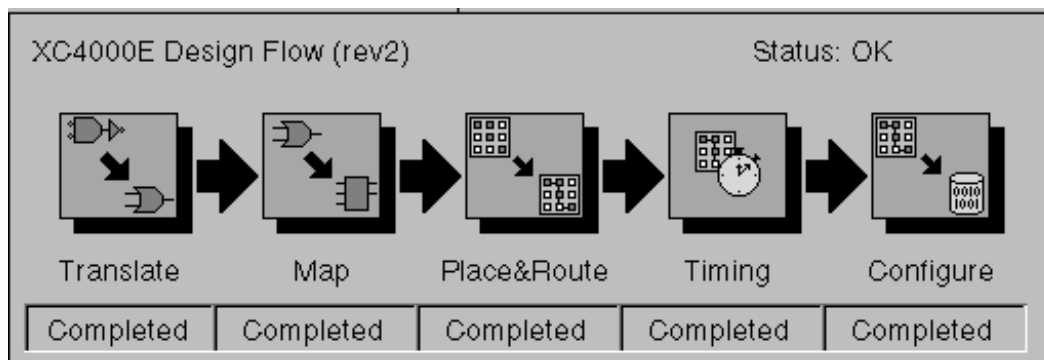


Figure 3-4 Flow Engine indicates completion of each design segment.

Configure

After place and route, the Flow Engine translates the physical implementation into a binary stream. The binary stream is used to program the FPGA. The binary stream is saved as a configuration file (.BIT) using the BITGen executable.

Analyzing Reports

The reports provide information on logic trimming, logic optimization, timing constraint performance, and I/O pin assignment. To access the reports, select from the Design Manager menu, Utilities->Report Browser. To open a particular report, double click on its icon. Refer to Figure 3-5.

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:

- Missing or untranslatable hierarchical blocks
- Invalid or incomplete timing constraints
- Output contention, loadless outputs, and sourceless inputs.

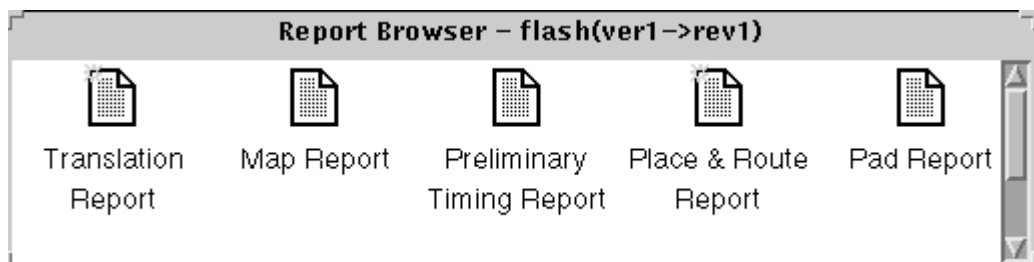


Figure 3-5 Report Browser

Map Report

The Map Report (.MRP) contains warning and error messages detailing logic optimization and logic mapping to physical resources. The report list the following:

- Removed logic - Sourceless and loadless signals can cause a whole chain of logic to be removed. Each deleted element is listed with progressive indentation, so the origins of removed logic sections are easily identifiable; their deletion statements are not indented.

- Added or expanded logic due to speed optimization.
- Design Summary lists the number and percentage of used CLBs, IOBs, Flip-Flops, and Latches. It also lists the use of architecturally specific resources like global buffers and boundary scan logic.

Note: The Map Report can be very large. To find information, use key word searches. To find sections, perform searches on '---', because each section heading is underlined with dashes.

Place and Route Report

The Place and Route Report (.PAR) contains the following information:

- Placer Score - The placer score measures the goodness of the placement. Lower is better. The score is strongly dependent on the nature of the design and the part targeted, so 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 Flowsw" at the end of this chapter.
- The timing summary at the end of the report details the designs asynchronous delays. For information on timing constraint performance and synchronous delays, refer to the Timing Analysis section later in this chapter.

Pad Report

The Pad Report lists the design's pinout in three ways:

- Signals referenced according to pad numbers
- Pad numbers referenced according to signal names
- PCF file constraints. This section can be cut and pasted into the .PCF file after the SCHEMATIC END; statement to preserve the pinout for future design iterations.

Selecting Options

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

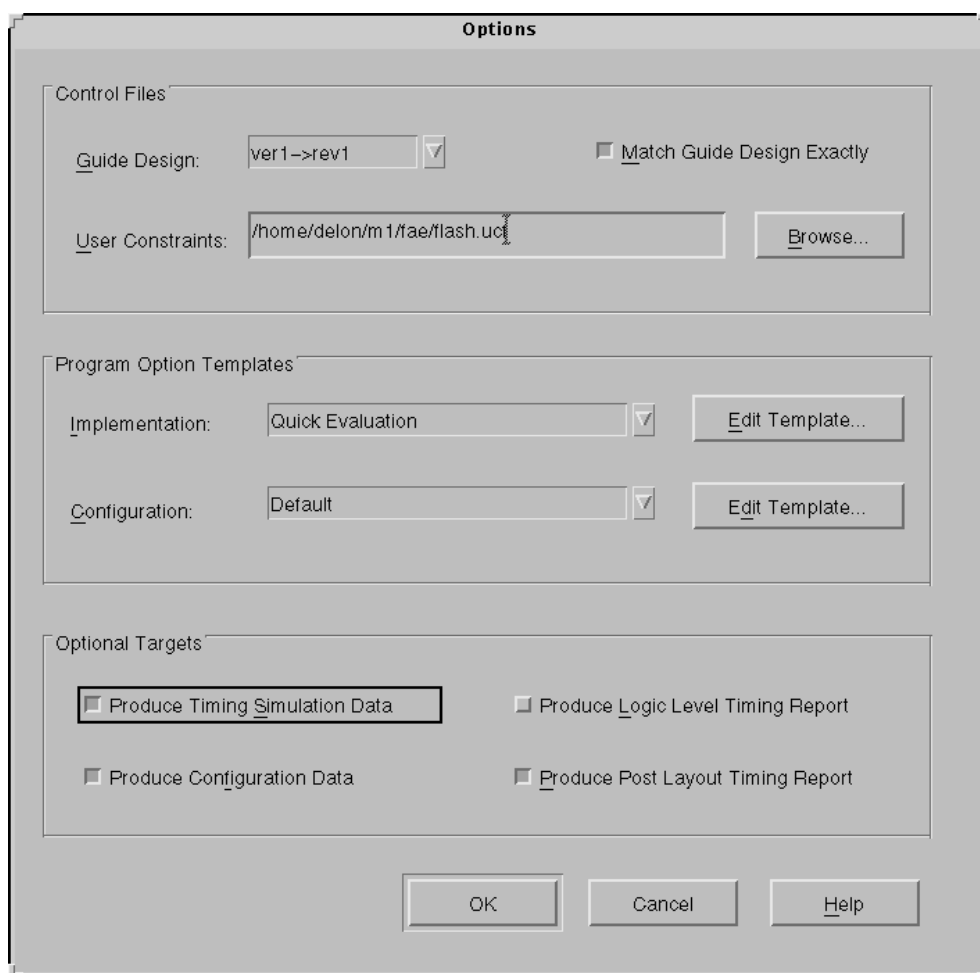


Figure 3-6 Options Dialog

You can have multiple templates in a project. By choosing a template, you are choosing an implementation or configuration style. To access the options and templates:

- Select the Options button in the Implement dialog, or from the Flow Engine menu select Setup->Options.
- In the Options Dialog, select the Edit Template button for Implementation or Configuration to access the associated template.
- From the Design Manager menu select Utilities->Template Manager

The default options settings should accommodate most implementations. For information on the options, select Help->Contents from the Design Manager menu.

Using Constraint Files

The M1 tools allow you to control the implementation of a design by entering constraints. There are two types of constraints that you can apply to a design: location constraints and timing constraints. Location constraints are used to control the mapping and positioning of the logic elements in the target device. The most common location constraints are pad constraints. They are used to lock the pins of the design to specific I/O locations so that the pin placement is consistent from revision to revision.

Timing constraints tell the software which paths are critical, and therefore need closer placement and faster routing. Timing constraints also tell the software which paths are not critical and therefore do not need closer placement or faster routing. Both the placer and the router can be timing constraint driven.

Design, Netlist, User, and Physical Constraints

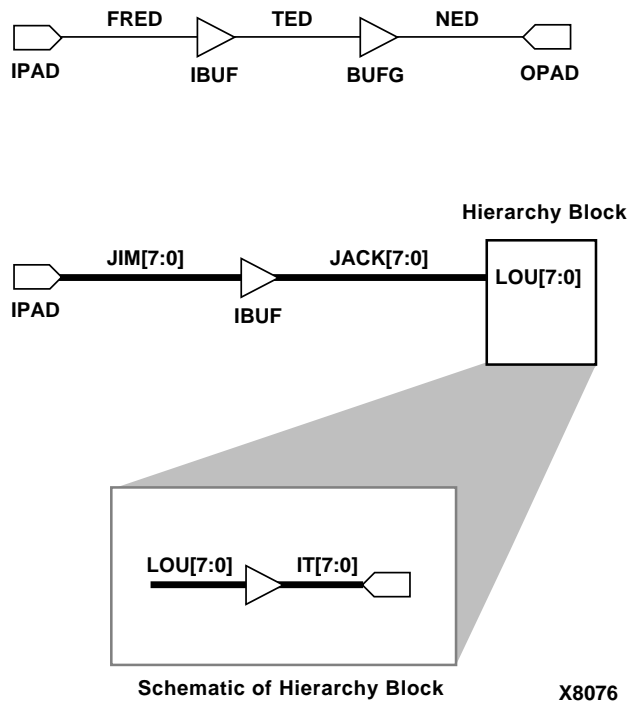
Constraints can be entered throughout the design and implementation processes. Constraints can be entered during the design phase by adding them to a schematic, specifying them to a synthesis tool, or listing them in a user constraint file. Constraints entered directly in the input design are known simply as design constraints and are ultimately placed in the design netlist. Constraints specified during a synthesis compilation result in a netlist constraints file `design_name.ncf`. If you want your constraints separated from the

input design files, or if you want to modify your constraints without having to completely re-synthesize your design, you can create a user constraints file `design_name.ucf`.

Creating A User Constraint File

The User Constraint File (.UCF) is a user-created ASCII file that holds timing and location constraints. It is read by ngdbuild during the Translate process, 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 then it will automatically be read. Otherwise, specify a file for User Constraints in the Options dialog.

The following example shows how to lock I/Os to pin locations, and how to write timespec and timegroup constraints.



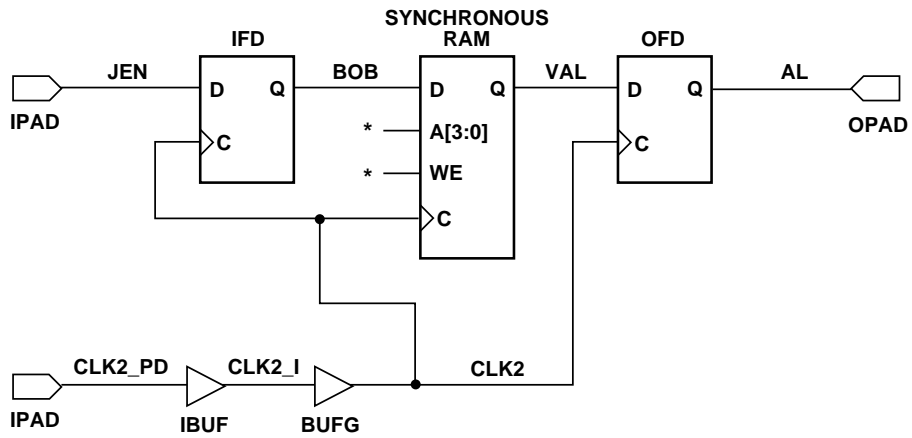
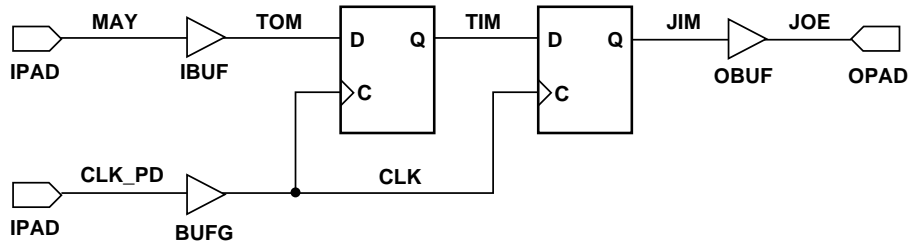
```
# This is a UCF comment
# The constraints below lock the I/O signals to pads
#The net name that connects to the pad is used to
constrain the I/O.
# The pin grid array packages use pin names like B3 or
T1, instead of P<Pin Number>.

# Lock the input pins
NET FRED LOC = P18;
NET JIM0 LOC = P20;
NET JIM1 LOC = P23;
NET JIM2 LOC = P24;
NET JIM3 LOC = P25;
NET JIM4 LOC = P26;
NET JIM5 LOC = P27;
NET JIM6 LOC = P28;
NET JIM7 LOC = P38;

# Lock the output pins
NET NED LOC = P19;
NET HIERARCHY_BLOCK/IT0 LOC = P44
NET HIERARCHY_BLOCK/IT1 LOC = P45
NET HIERARCHY_BLOCK/IT2 LOC = P46
NET HIERARCHY_BLOCK/IT3 LOC = P47
NET HIERARCHY_BLOCK/IT4 LOC = P48
NET HIERARCHY_BLOCK/IT5 LOC = P49
NET HIERARCHY_BLOCK/IT6 LOC = P50
NET HIERARCHY_BLOCK/IT7 LOC = P462
```

For more information on constraint precedence, refer to the
“Libraries Guide”.

The second example shows how to specify timing constraints.



* Nets not used in timing constraints.

X8075

---User Constraint File (UCF):

```
# This is a comment
# Period specifies minimum PERIOD of CLK net. Offset
# specifies that data on MAY can arrive up to 6 ns
# before the clock edge arrives on CLK.
# NOTE: Period constraints do not apply to elements in
# output pads.
NET CLK PERIOD = 20 ns ;
NET MAY OFFSET = IN : 6ns : before : CLK_PD ;
```

```
# Groups all clocked loads of CLK2 into CLK2_LOADS
# timegroup
# Groups all clocked loads of VAL into VAL_LOADS
# timegroup TNM # => Timegroup NaMe

NET CLK2 TNM=CLK2_LOADS ;
NET VAL TNM=VAL_LOAD ;

# Specifies worst case speed of path from IPAD to CLK2
# loads.
# Includes pad, buffer, and net delays. TS01 is a
# timespec identifier; it # can have names of the form
# TS<string>. PADS (CLK2_PD) is a
# timegroup name specified inside of a timespec.
TIMESPEC TS01=FROM : PADS (CLK2_PD) : TO :
    CLK2_LOADS=15ns ;

# Specifies the maximum frequency for all loads
# clocked by CLK2.
TIMESPEC TS02=FROM : CLK2_LOADS : TO :
    CLK2_LOADS=30Mhz;

# Specifies the minimum delay on the path from
# Synchronous
# RAM to OFD. Includes clock to Out delay, net delay,
# and setup time.
TIMESPEC TS03=FROM : CLK2_LOADS : TO :
    VAL_LOAD+15000ps ;
```

Guiding An Implementation

In a design process a design is modified and implemented many times. The changes are such that from one implementation to the next there are parts of the design that do not change. Guiding a design accelerates iterative implementations by reusing the unchanged sections from a previous implementation on current implementations. The software therefore only has to spend time generating implementations for sections of the designs that have changed. In the M1 tools guiding is used during map, place, and route. Guiding a design can significantly reduce run times, since less processing has to occur

To select a previous implementation to guide a current implementation, open the Options dialog. In the Guide Design dropdown box,

you can select previously implemented revisions, Project Clipboard, Custom, or None. The Project Clipboard is used to save the guide data of revisions that are being overwritten. Guide data can be saved to the Project Clipboard by selecting the Copy <previous revision> guide data to project clipboard button in the Implement dialog. NCD files created outside of the project can be used for guiding by selecting the Custom option. In the Custom dialog pop-up, be sure to enter a mapped ncd file, and a placed and routed ncd file. If guide files aren't needed, select None.

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, such as re-synthesis of an entire hierarchical block.

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

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 block delays on timing constraints, clock frequencies, and path delays. A post-place-and-route timing report, that incorporates both block 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. Although route delays are not accounted for, the logic delays can provide valuable information about the design.

If logic delays account for a significant portion ($> 50\%$) for the total allowable delay of a path, the path may not be able to meet your timing requirements once 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% to 60% 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 less levels of logic, tag the paths for specialized routing resources, move to a faster device, or allocate more time for the path.

If logic-only-delays account for much less ($<15\%$) than the total allowable delay for a path or timing constraint, then very low placement 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 all 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 placer 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, or allocate more time for the paths.

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: 1) edit the template implementation, 2) select the timing tab, and 3) select a report format.
- Once MAP or PAR has completed, the respective timing reports will appear in the report browser.

Detailed Timing Analysis

To perform detailed timing analysis, select Tools->Timing Analyzer from the Design Manager menu. 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., 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.

When Can Simulation Data Be Created

The M1 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, all net delays are set to 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

To create timing simulation data:

- Open the Options dialog and select the Produce Timing Simulation Data option.
- In the same dialog, click on the Edit Template button for Implementation.
- In the Implementation Template dialog, select the interface tab.

- Select the desired simulation netlist format: EDIF, VHDL, Verilog, or XNF.
- If you select EDIF, choose a Vendor: Generic, ViewLogic, Mentor, or LogicModeling.
- Select Correlate Simulation Data to Input Design if you are using a simulation stimulus file or test fixture that was used for functional simulation, or that has signal names that were optimized out of the design during implementation.

With these options selected, the Flow Engine will automatically create a post-route simulation netlist of the format you have selected, during the Timing stage. To access the simulation netlist, in the Design Manager select the revision and from the menus choose Design->Export. In the Export dialog, select Timing Simulation Data and the directory you want the file exported to. When you hit OK, the listed netlist will be copied to the selected directory. Use the netlist as an input to your simulator to perform a timing simulation.

Note: For information on NGDAnno, 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`, `NGD2EDIF`, or `NGD2NNF`. The following commands create a functional simulation netlist.

```
ngdbuild design_name
ngd2edif design_name
```

`NGD2XNF` can be used to create a Xilinx Netlist Format file containing version 16.0 XVF primitives. This file can then be used by any third part simulator, which does not yet support EDIF, VHDL, or Verilog formats.

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 sections shows the most common strategies.

Re-Entrant Route

PAR can take an implemented design as an input, and re-route it. 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. As long as an NCD file exists that is at least placed, PAR can use it for re-entrant routing. To execute re-entrant route:

- In the Design Manager, select the implemented revision, and select the Flow Engine button in the toolbox
- In the Flow Engine, select the Setup->Advanced menu
- In the Advanced dialog, select the Allow Re-Entrant Route button, which enables the re-entrant route options.
- If meeting timing specifications is a critical goal for the route, then select the Use XACT Timespecs button. If meeting timing specifications is not critical, then do not select the button because timing driven route takes much longer to process than non-timing driven route.
- Cost based routing, delay based routing, and generic routing are three routing styles that use different algorithms. The effectiveness of each style is strongly dependent on the design and part, so no predictable criteria can be suggested to choose one style over others. The best methodology is to select less than three passes for each. If the design still does not meet your requirements, use the Place and Route report to determine which style was most effective and try using more passes of that style.
- Click on OK on the Advanced dialog to submit the options. This causes the Place and Route graphic in the Flow Engine to show 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 M1 software on networked UNIX workstations, then to significantly reduce run time, the place and route passes can be run in parallel, by executing each pass on a separate machine. To execute Multi-Pass Place and Route:

- In the Design Manager, be sure to select a version and not a revision, and then from the menu choose Design->FPGA Multi-Pass Place and Route.
- In the FPGA Multi-Pass Place and Route dialog, select a value for the Starting Strategy. The Starting Strategy 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 Starting Strategy to 1, since 0 was used in your previous single-pass run.
- Select the number of iterations to run.
- Select the number of iterations to save. Based on the design score, only the files from the best runs are saved. If you are running on a UNIX workstation and want 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.
- Click OK to launch the Multi-Pass Place and Route Process.

Appendix A

Cadence Concept and Verilog Interface Notes

Introduction

This appendix covers how to set 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.

Documentation

The following documentation is available for the Cadence interface:

- Cadence Interface/Tutorial is available on the CDROM supplied with your software.
- The Cadence software documentation (for Cadence applications such as Concept and Verilog-XL) is available only from Cadence and is viewable by entering “openbook” on the UNIX command line.
- The M1 Software Release Notes describes installation, setup, and any current issues regarding the use of the Cadence interface.

Setting up the Xilinx/Cadence Interface

In addition to the environment variables discussed in the Chapter 1, the following environment variables must be modified or added to run the Xilinx/Cadence interface tools:

- CDS_INST_DIR (add for Concept)
- VERILOGEXE (add for Verilog-XL)
- XAPPLRESDIR (add for Verilog-XL)

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

These variables should be set in the following manner:

```
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

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

set path = ( $XILINX/cadence/bin/<platform_name>
$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 SunOS and 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
```

Note: It is common for users 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), “sun4” (for SunOS), “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” above.

For example:

```
setenv CDS_INST_DIR /products/cds.ver9604

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

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

setenv LD_LIBRARY_PATH $CDS_INST_DIR/tools/lib:
$OPENWINDHOME/lib : /usr/lib : /tools/x11r5/sun4/lib:
$LD_LIBRARY_PATH

set path = ($XILINX/cadence/bin/sun \
           $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 above settings assume that the **XILINX** and **LD_LIBRARY_PATH** environment variables point to the appropriate areas.

Cadence/Verilog and M1 Design Flow

The design flow (refer to Figure A-1) shows how Cadence Concept and Verilog-XL interact with the Xilinx M1 Software. The design flow shows design entry, functional simulation, implementation, and timing simulation.

- The design is first 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, if the design is purely schematic after analyzing it with Concept 2XIL.

Cadence

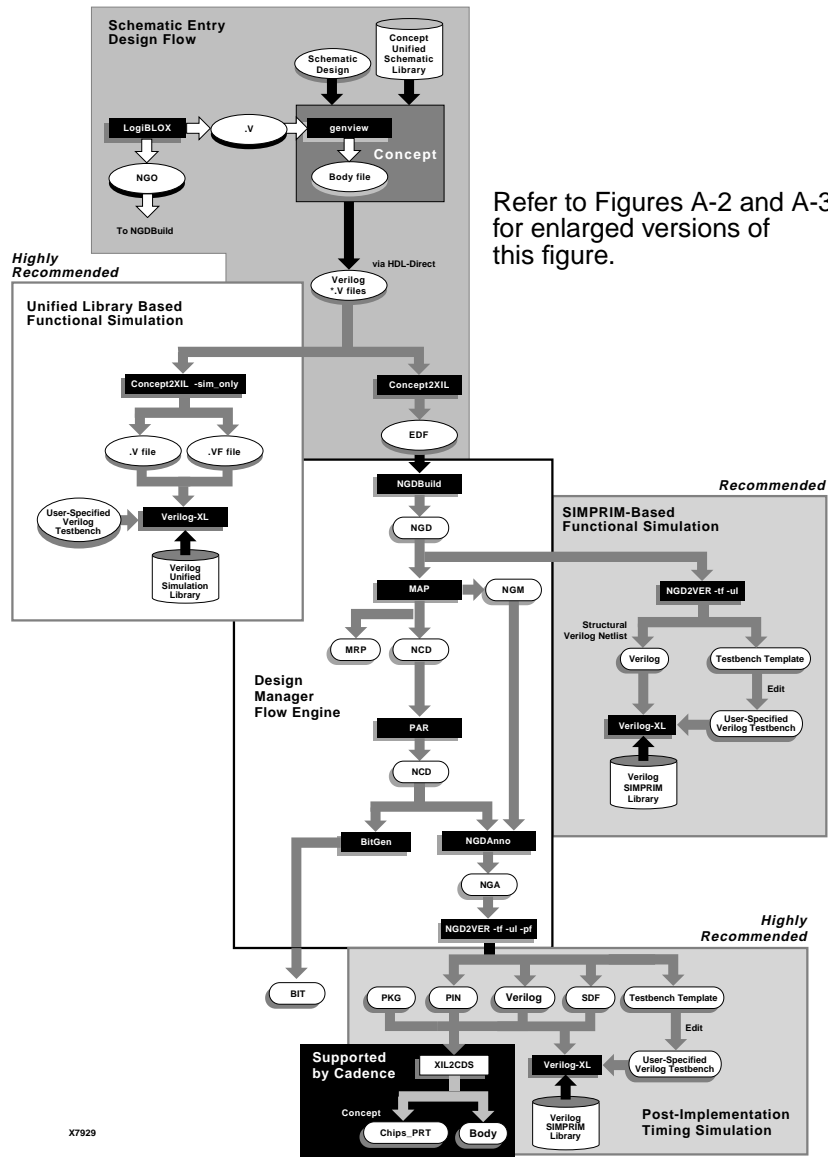


Figure A-1 Cadence/Verilog Interface and M1 Design Flow

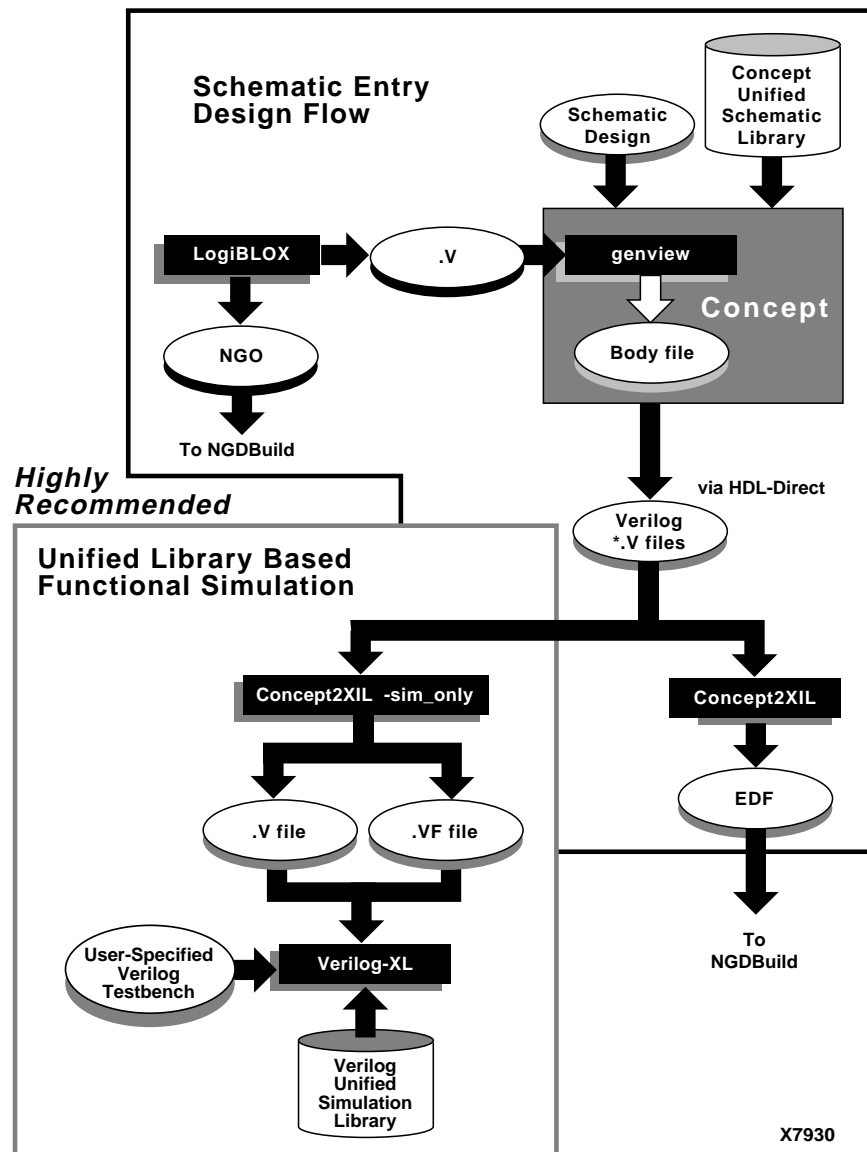
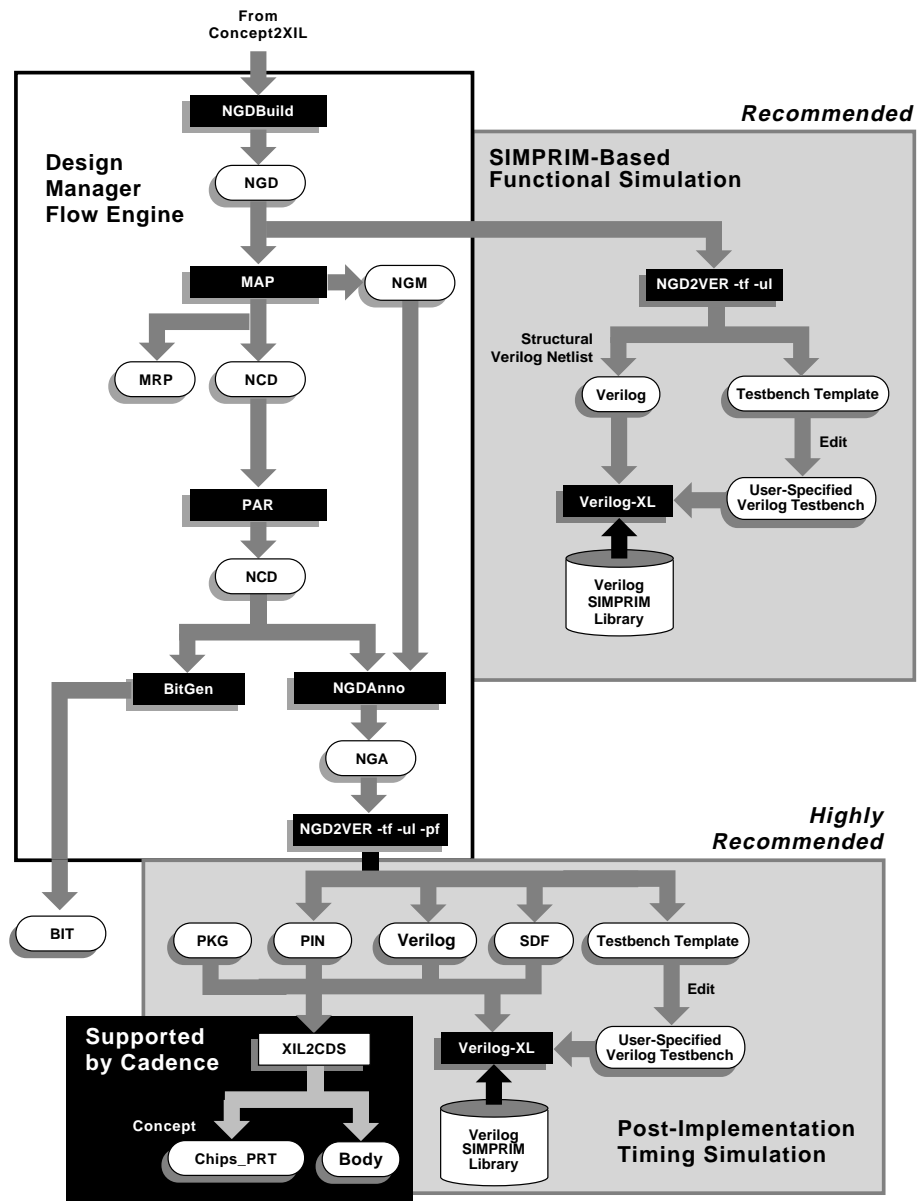


Figure A-2 Top Portion of Figure A-1



X7931

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 functional simulation was not performed before NGDBUILD (if the design contains non-schematic blocks), then functional simulation may be performed after NGDBUILD is completed. All non-schematic blocks, if any exist, are merged in by NGDBUILD, so all designs can be simulated at this point.
- The EDF file is passed to the Xilinx M1 Core Technology tools (Design Manager Flow Engine) for implementation.
- The Xilinx core 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

The Xilinx/Concept interface requires that three files be present in the design directory. The following is a brief discussion of those files (global.cmd, master.local, and cds.lib). Refer to the Cadence Interface/User Guide (located on the CDROM supplied with your software) for more information.

- **global.cmd**

You will need a global.cmd that references the proper libraries. Here is an example global.cmd:

```
master_library "./master.local" ;
library "xce4000ex" ,
        "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 family you are using. In this example, the “xce4000ex” alias points to the XC4000EX 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**

This 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 a 4000EX design:

```
file_type = master_library;

"xce4000ex" '/xilinx/cadence/data/xce4000ex/ \
  xce4000ex.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**

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 4000EX design:

```
define xce4000ex_syn /xilinx/cadence/data/ \
  xce4000ex_syn
```

The format for entries in this file is:

```
define <target_tech>_syn  <path_to_XILINX>/cadence/data/ \
  <target_tech>_syn
```

where *target_tech* is xce3000, xce4000e, xce4000ex, xce5200, xce7000, or xce9000.

Using HDL Direct

The M1 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
runopl <installation_path_to_cadence>/tools/fet/concept/
      hdl_direct/bin/autosym
```

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 Vs. Size Support

The M1 Software does 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 Up Concept

To start the Concept editor, type:

```
concept &
```

Pure Concept Schematic Functional Simulation

It is possible to functionally simulate your design before translating the design, if the design is purely schematic (if there are no “black boxes” in the design). Assuming that HDL Direct was on 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 xce4000ex my_design
```

Go to the xilinx.run directory, and create a test fixture file in a text editor.

```
verilog +delay_mode_unit my_design.v -f my_design.vf  
my_test fixture.tv
```

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

Translating a Design to Xilinx EDIF

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

```
concept2xil -family target_tech design_name
```

For example, to target my_design to the XC4000EX:

```
concept2xil -family xce4000ex 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 M1.0 Concept libraries (xce***) where *** = target architecture.

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:

```
ngdbuild -p part_name design_name
```

```
ngd2ver -ul -tf design_name.ngd
```

```
verilog +delay_mode_unit design_name.v glbl.v test_fixture.tv
```

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 my_design.nga
```

This creates a `.V`, `.SDF` and `.TV` file. The `-tf` option causes NGD2VER to automatically create a test fixture template file, which is named `my_design.tv`. Edit the `.TV` file to add the appropriate stimuli. The `-ul` option causes NGD2VER to automatically add a “uselib” directive to the `.V` file that references the Xilinx-supplied Verilog SIMPRIM libraries. 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.

```
verilog my_design.v glbl.v my_design.tv
```

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:

```
xil2cds <routed_design> -lwbverilog  
-use <name_of_.wrk_file> -r <run_directory>  
-family xce4000ex -mode all
```

Example: (design name is “`my_design_r`”, `.WRK` file is `design.wrk`, run directory is the current directory, architecture is XC4000EX, `-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 xce4000ex -mode allXIL2CDS 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 on 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 (e.g., IBUF8), then you must write the pin constraints in 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.

To add a LOC property:

1. Enter the “Attribute” mode, and select the IBUF/OBUF to LOC.
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 is RC, where R is the actual row and C is the actual column of the device pin. For example: LOC=A13.

3. Select Done. You may reposition the LOC property above the PAD, if you wish, 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 (e.g., “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 \$XILINX/cadence/data/xilinx.pff to your design directory, and add entries for other labels. For more information on this subject, please refer to the *Cadence Interface/Tutorial Guide*. For more information on timing constraints, see the XACT-Performance Utility chapter of the *Development System Reference Guide*.

Appendix B

FPGA Express Interface Notes

Introduction

This appendix covers how to install and start using FPGA Express and the XACTstep vM1.x.x Pre-Release. Synopsys and the Xilinx CDROM documentation are referenced to assist the user in finding further information.

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 which is fully compatible with XACTstep vM1.x.x. Only the core tools and a third party simulation tool are needed in addition to FPGA Express to fully create and simulate a design.

Installation of FPGA Express

Insert the FPGA Express CD into your CDROM drive. Start the Explorer and double-click on the CDROM icon. Double-clicking on the setup.exe “application” starts 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.

Design Entry With FPGA Express

To complete a design entry, proceed with the following steps.

1. Start FPGA Express. In Windows 95 and Windows NT, FPGA Express can be started from the “Start Bar”. Click on Program -> Synopsys -> FPGA Express.

2. Enter your design as Verilog or VHDL files, using a text editor of your choice.
3. Define your project in FPGA Express. Go to File->New Project.
4. Tell FPGA Express which file in your project is the top-level file. Select the top-level file in the "<top level design>" drop-down list, which is in the middle of the FPGA Express toolbar.
5. Create an implementation. Go to Synthesize->Create Implementation.
6. Optimize the design. Note: this is the actual synthesis procedure. Go to Synthesize->Create Implementation.
7. Write out a XNF file. Go to File->Export Netlist.

The XNF file can now be given to the M1 core tools or Design Manager.

The FPGA Express design flow takes in Verilog or VHDL and outputs a XNF file, which can be processed directly by the M1 core tools or the Design Manager. 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.

Simulation With FPGA Express

FPGA Express is a synthesis tool only. Simulation of designs with FPGA Express must be done with a third part simulation tool. For more information on simulation with FPGA Express, refer to the documentation of your third party simulation tool. Note: There are four types of simulation possible: behavioral, post-ngdbuild, post-synthesis functional, and post-synthesis post-route timing simulation.

Documentation

The following documentation is available for FPGA Express and the XACTstep vM1.x.x XC4000EX Pre-Release:

- For installation of XC4000EX Pre-Release core tools and/or GUIs, refer to XC4000EX release document

- 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 M1 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 file.

FPGA Express/XC4000EX Pre-Release Design Flow

FPGA Express is intended to be the top-level design tool in the design flow. FPGA Express writes out a XNF file which is fully compatible with XACTstep v M1.x.x. The XNF file written out by FPGA Express can be accepted by ngdbuild or the Design Manager.

In designing with FPGA Express, there are four types of simulation possible:

1. behavioral
2. post-ngdbuild
3. post-synthesis functional
4. post-synthesis post-route timing simulation

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

Refer also to "FPGA Express/M1 Design Flow", Figure B-1.

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, the user may write the constraints in a .UCF file. The timespecs created by FPGA Express in the XNF file have the FROM:TO syntax.

Porting Code from FPGA Compiler to FPGA Express

Read only this section if you are compiling 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 M1 XSI or XACT XSI libraries, some of these components do not exist in the FPGA Express libraries.

Some of the components that can be instantiated in the M1 Software M1.x.x flow cannot be instantiated in the FPGA Express tool, since there are slight differences in names. For example, the BUFGP_F which exists in the XSI component library doesn't 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

fpgaexpress is the directory where FPGA Express has been installed on your system. Inside each of the above, there are files with the three-character extension *.dsn*. The string in front of *.dsn* is the name of the CELL that can be instantiated in FPGA Express.

In general, instantiation will not be necessary. For the 4000EX FPGA Express flow, the following five items must be instantiated:

I/O muxes	fast capture latches
RAM	BSCAN
LogiBLOX	

For examples of instantiation of I/O muxes, fast capture latches, RAM, and BSCAN, please refer to "Entity Coding Examples" in the appendix entitled "Synopsys Interface Notes", in this manual.

Note: Refer to Figure B-1. For further information on NGDbuild, Design Manager, MAP, NGDAnno, NGD2VER, and/or NGD2VHDL, refer to the "Design Manager/Flow Engine/User Guide" and to the "Development System Reference Guide".

FPGA Express

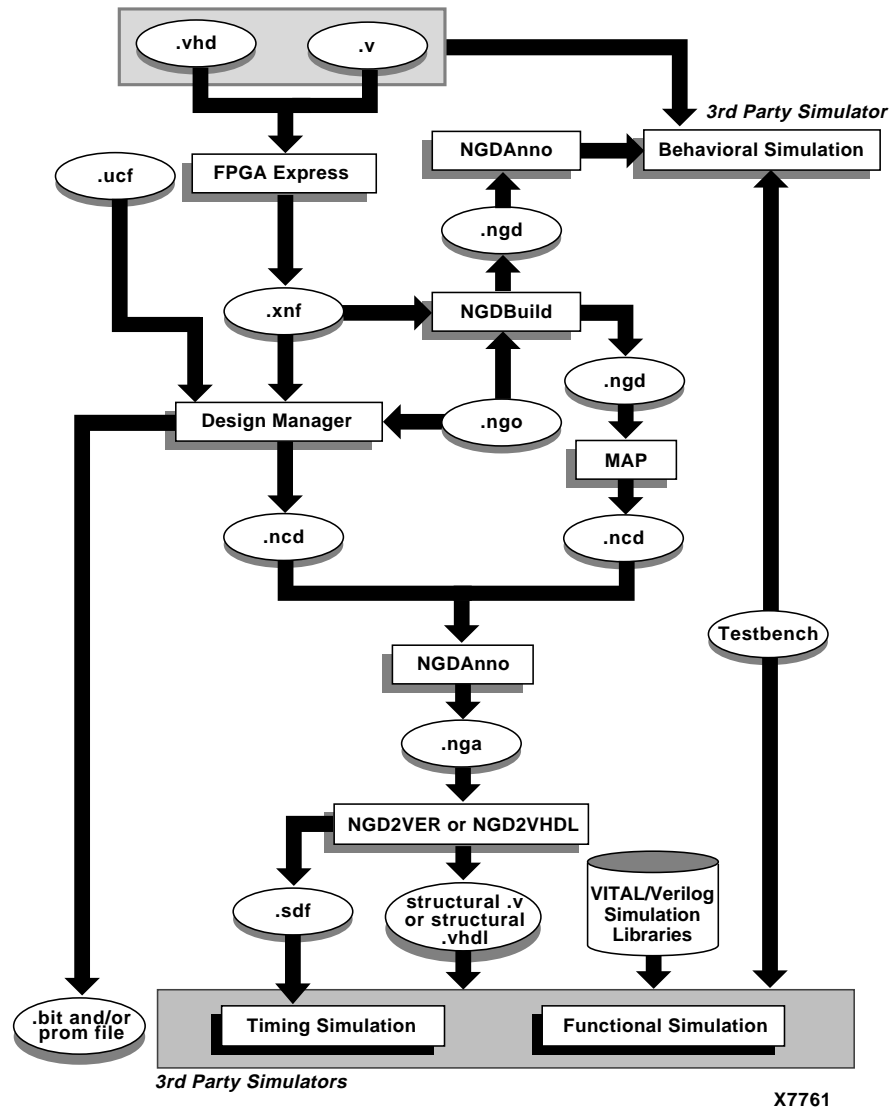


Figure B-1 FPGA Express/M1 Design Flow

Appendix C

Mentor Graphics Interface Notes

Introduction

This appendix covers how to set up the Mentor Graphics interface and associated libraries. You will be guided through a brief illustration on how to lock pins and enter timing constraints.

Documentation

The following documentation is available for the Mentor Graphics interface:

- Mentor Graphics Interface/User Guide is available on-line and viewable with a DynaText browser.
- Mentor Graphics software documentation (for applications such as Design Architect, QuickSim, QuickHDL, and DVE) is available on-line and viewable with the Mentor-supplied BOLD Browser.
- M1 Software Release Notes which describe 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 Chapter 1, the following environment variables must be modified or added to run the Xilinx/Mentor interface tools:

- MGC_HOME (add)
- LCA (add)
- MGC_GENLIB (add)
- MGC_LOCATION_MAP (add)

- path (modify)
- LD_LIBRARY_PATH (modify for OSunOS/Solaris)
- SHLIB_PATH (modify for HP-UX)

These variables should be set in the following manner:

```
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 SunOS and 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
```

For example:

```
setenv MGC_HOME /usr/mentor
setenv LCA $XILINX/mentor/data
setenv MGC_GENLIB $MGC_HOME/gen_lib
setenv MGC_LOCATION_MAP /usr/data/mgc_location_map
set path = ( $XILINX/mentor/bin/sun $path)
setenv LD_LIBRARY_PATH $MGC_HOME/shared/
lib:$MGC_HOME/lib:$LD_LIBRARY_PATH
```

Note: The above settings assume that the Xilinx environment variables point to the appropriate area, as described in the Chapter 1 of this manual.

Mentor/M1 Software Design Flow

Refer to Figure C-1 and the design flow of the Mentor Graphics tools and the M1 Software tool. Shown are design entry, functional simulation, implementation, and timing simulation.

- At the top, the design flow starts with the design being entered into PLD_DA (the Mentor schematic design tool, also known as EDDM netlist).
- 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 M1 Software Core Tools for implementation.
- The Xilinx core tools create an EDN file which is then processed by PLD_EDIF2TIM to get a timing-annotated EDDM netlist.
- This new netlist is processed by PLD_DVE to generate a Xilinx-style design viewpoint.
- The design is then passed to PLD_Quick Sim to run in cross-probing mode for timing simulation.

For functional simulation, first generate a simulation viewpoint, which can be done with PLD_DVE. For example, to generate a viewpoint for the XC4000EX component my_design:

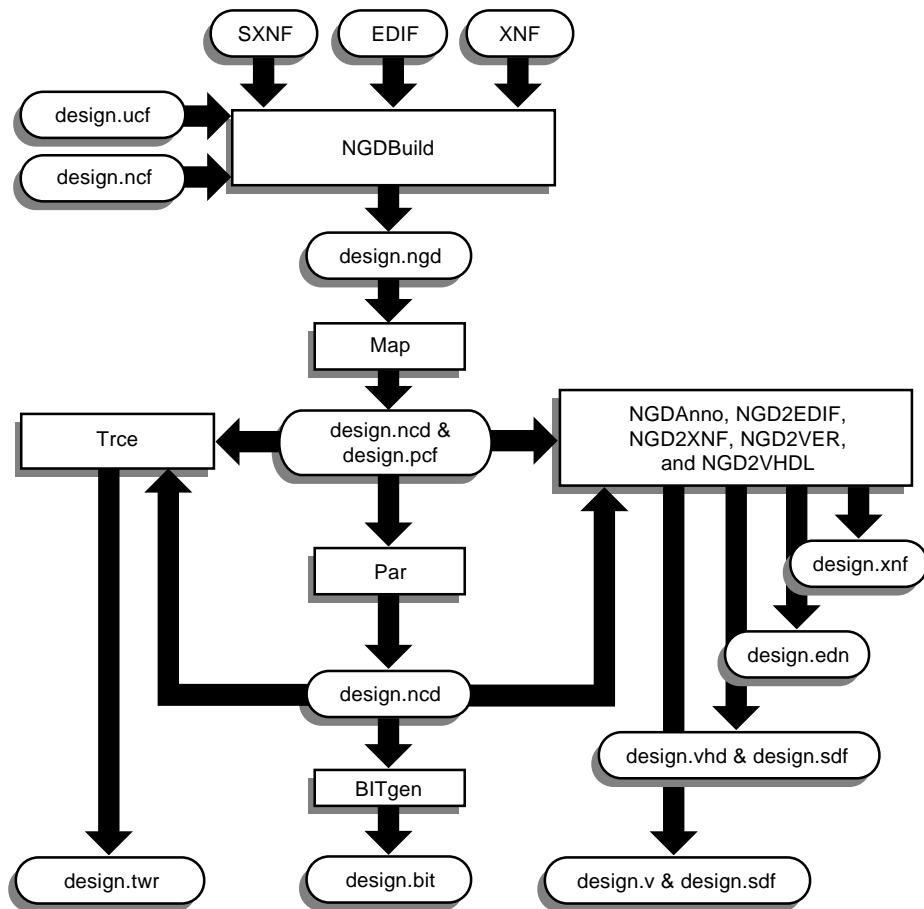
```
pld_dve my_design xc4000ex
```

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

To simulate this design, run:

```
pld_quicksim my_design -noc
```

This runs QuickSim for functional simulation without cross-probing.



X8037

Figure C-1 Mentor/M1 Software Flow

You may also use the PLD_DVE and PLD_QuickSim icons in PLD_DMGR. For more information on functional simulation, see the “Functional Simulation” chapter or the “Manual Translation” chapter of the *Mentor Graphics Interface/User Guide*.

Translating a Design to Xilinx EDIF

To translate a design into an EDIF file for the Xilinx core 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. 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 “Design Implementation” chapter or the “Manual Translation” chapter of the *Mentor Graphics Interface/User 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.

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.

Creating a Simulation Viewpoint

You must create a simulation viewpoint for this component.

For example, to create a simulation viewpoint for the timing model `my_design`, enter the following:

```
p1d_dve my_design_lib/my_design xc4000ex
```

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 add the `-noc` option to run without cross-probing.)

```
p1d_quicksim my_design_lib/my_design
```

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 “Timing Simulation” chapter or the “Manual Translation” chapter of the *Mentor Graphics Interface/User Guide*.

Mentor-Related Environment Variables

The M1 Software Mentor Interface requires the setting of 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 “sun” for SunOS workstations, or with “hp” for HP-UX workstations.) These variables are in addition to both the XILINX environment variable settings required by the core tools (referred to in Chapter 1 of this manual) and to the Mentor-specific variables such as `MGC_HOME` and `MGC_LOCATION_MAP`. See the Mentor Graphics documentation for more information on the latter.

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, please see your Mentor Graphics documentation.

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 (e.g., 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 “Attributes, Constraints, and Carry Logic” chapter of the *Libraries Guide*.

Timing Constraints

Timing constraints may 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 (e.g., “FROM:FFS:TO:FFS=30NS”) is entered as the property value. For more information on timing constraints, see the “XACT-Performance Utility” chapter of the *Development System Reference Guide*.

Appendix D

Synopsys Interface Notes

Introduction

This appendix covers how to set up the Synopsys interface and associated libraries. Example files are included to help you set up the FPGA Compiler and VSS with the M1 Software.

Documentation

The following documentation is available for the Synopsys interface.

- The Synopsys (XSI) Interface/Tutorial Guide is available on the CDROM included with your software package.
- The M1 Release Notes describe installation setup and current issues regarding the use of the Synopsys interface. The “Release Notes” is included with your software.
- For converting an XACT 5.x.x Synopsys design to M1, refer to the Xilinx Software Conversion Guide.

Setting Up the Xilinx/Synopsys Interface

In addition to the environment variables discussed in Chapter1, the following environment variables must be modified or added to run the Xilinx/Synopsys interface tools:

- SYNOPSYS (add)
- PATH (modify)
- LD_LIBRARY_PATH (modify)
- SHLIB_PATH (modify)

These variables should be set in the follow manner:

```
setenv SYNOPSYS <installation_path_to_synopsys>
set path = ( $XILINX/synopsys/bin/<platform_name> \
             $SYNOPSYS/<platform_name>/syn/bin    \
             $SYNOPSYS/<platform_name>/sim/bin    \
             $path)
```

For SunOS and 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
```

For example:

```
setenv SYNOPSYS /usr/synopsys
set path = ( $XILINX/synopsys/bin/sun \
             $SYNOPSYS/sun/syn/bin    \
             $SYNOPSYS/sun/sim/bin    \
             $path)

setenv LD_LIBRARY_PATH $SYNOPSYS/sun/sim/
lib:$LD_LIBRARY_PATH
```

Note: The above settings assume that the Xilinx environment variable points to the appropriate area, as described in Chapter 1.

Synopsys/M1 Software Design Flow

The flow diagram in Figure D-1 illustrates the synthesis, implementation and simulation design flow through Synopsys and Xilinx M1 Software. Below is a brief description of the flow.

Inputs to both the FPGA Compiler and the Design Compiler are:

- Synthesis Script (DC Script and FPGAC Script in Figure D-1)
- Source HDL (VHDL or Verilog)
- **.synopsys_dc.setup** (Synopsys setup file)

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*.sxnf design netlist written by the FPGA Compiler in XNF format.
- *design_name*.sedif design netlist written by the Design Compiler in EDIF format.
- For more information, see the Synopsys (XSI) Interface/Tutorial Guide.

Examples of Synopsys Setup Files

The following section outlines the types of Synopsys setup files required to operate the FPGA Compiler and VSS correctly with the M1 Software tools. These examples are for targeting an XC4000EX device. Other FPGA and CPLD templates may be found in the Xilinx installation path, \$XILINX/synopsys/examples.

.synopsys_dc.setup

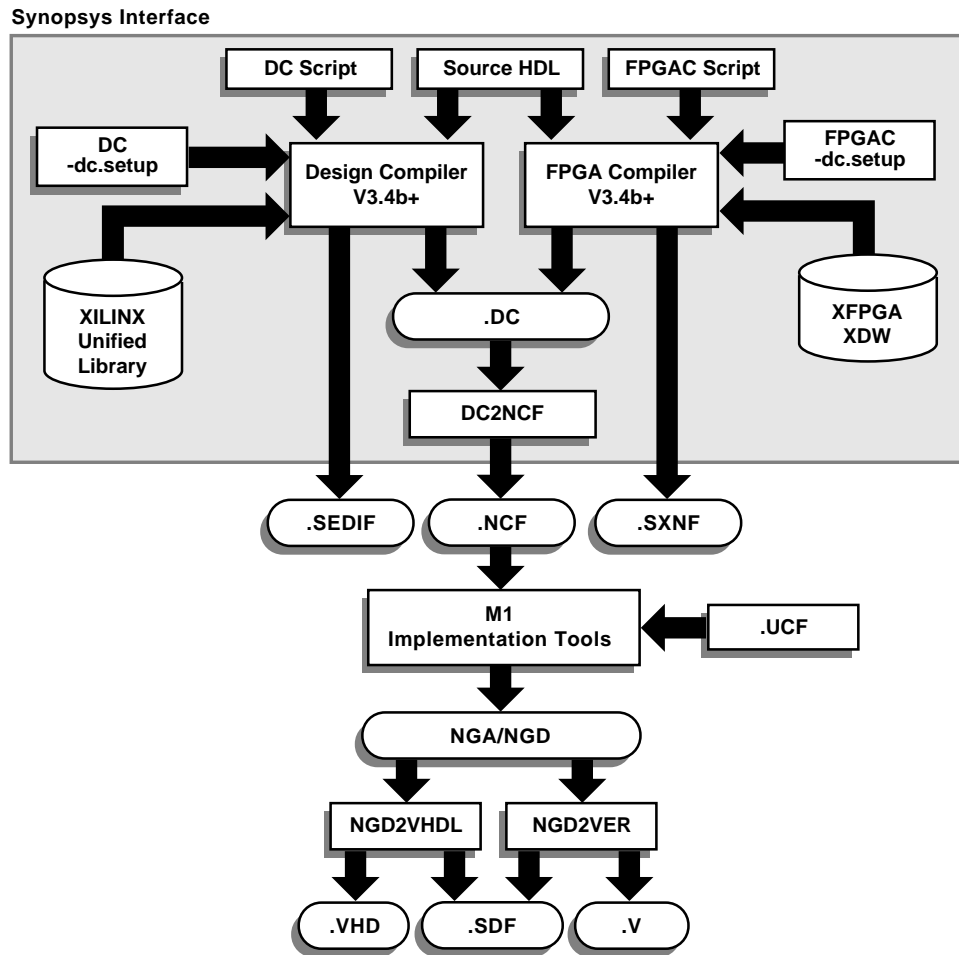
```
/* Template .synopsys_dc.setup file for Xilinx      */
/* For targeting a XC4000EX */

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_4000ex -path \
    XilinxInstall + /synopsys/libraries/dw/lib/xc4000ex
```



X8040

Figure D-1 Synopsys/M1 Software Design Flow

```

/* 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    */

```

.synopsys_vss.setup

```

/*      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 LogiBLOX simulation libraries            */
SIMPRIM : $XILINX/synopsys/libraries/sim/lib/simprims
LOGIBLOX : $XILINX/synopsys/libraries/sim/lib/logiblox
/* Example pointers to the Xilinx FTGS simulation */
XC3000A : $XILINX/synopsys/libraries/sim/lib/xc3000a/ftgs
XC4000E : $XILINX/synopsys/libraries/sim/lib/xc4000e/ftgs
XC5200 : $XILINX/synopsys/libraries/sim/lib/xc5200/ftgs
XC7000 : $XILINX/synopsys/libraries/sim/lib/xc7000/ftgs
XC9000 : $XILINX/synopsys/libraries/sim/lib/xc9000/ftgs

```

Example Script File

The following section describes the typical sequence of commands used to process designs with Synopsys. The commands are intended to be executed at the dc_shell command line, either individually or in “batch” mode. While the specific nature of a particular design may make some of the indicated commands unnecessary, or require the

addition of extra processing steps, the example below represents a good starting point for most designs.

Note that the script file includes examples illustrating techniques such as: 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 -slebrate 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
/* 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
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. */
```

Timing Constraints and DC2NCF

Timing constraints issued to Synopsys to control the synthesis process can be carried forward to the design implementation tools where they similarly 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 a design within the Synopsys environment to equivalent constraints that control the Xilinx place and route process. The automatic translation of these constraints is convenient because it relieves the need to apply the constraints twice (once for Synopsys and again for Xilinx) and ensures that the constraints used by Xilinx are equivalent to those applied to Synopsys.

DC2NCF supports translation of the following timing constraint commands from within Synopsys:

- create_clock
- set_input_delay
- set_output_delay
- set_max_delay
- set_false_path

The presence of other Synopsys timing constraint commands in a Synopsys script file will result in a warning from DC2NCF and no translation of that unsupported constraint will occur.

DC2NCF Design Flow

The Synopsys user is expected to validate the timing constraints by first constraining their design and compiling it. Having compiled their design (and in the case of XC4000E/EX FPGA Compiler users, performed the replace_fpga command), the design should be written

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... menu pick.

FPGA Compiler Users

Before writing either the netlist or the constraints file, any hierarchy in the design should first be flattened. While flattening a design's hierarchy removes hierarchy information from Synopsys's 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:

```
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

VHDL Code

```
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
component RAM32X1S
port(O: out STD_LOGIC; D: in STD_LOGIC;
A4,A3,A2,A1,A0,WE,WCLK: in 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 ILFFX
port(Q: 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: ILFFX port
map(Q=>LTCHOUT,D=>LTD,GF=>LTGF,CE=>LTGE,C=>LTCLK);
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,LTC
LK;

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));

ILFFX U3 \
(.Q(LTCHOUT),.D(LTD),.GF(LTGF),.CE(LTGE),.C(LTCLK);

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

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

endmodule
```

Comments About Code

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

Appendix E

Viewlogic Interface Notes

Introduction

This appendix covers how to set up the Viewlogic interface and project libraries. Included are examples for assigning location constraints and for using special XC4000EX features.

Documentation

The following documentation is available for the Viewlogic interface.

- “Viewlogic Interface/User Guide” is available on-line and viewable with a DynaText browser.
- “M1 Software Release Notes”, which describes installation setup and current issues regarding the use of the Viewlogic interface, is available on the supplied CDROM.

Setting Up Viewlogic Interface on Workstations

In addition to the environment variables discussed in the chapter entitled “Design Tools Setup”, the following environment variables must be modified or added to run the Xilinx/Viewlogic interface tools:

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

- SHLIB_PATH (modify)

In addition to the variables set for the Xilinx Core Technology tools, these variables should be set in the following manner:

```
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 SunOS and 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 above 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, as described in Chapter 1. The **POWERVIEW** variable is not required by Xilinx nor by Viewlogic software. It is used only to simplify these environment variable settings.

Setting Up Xilinx/Viewlogic Interface on the PC

In addition to the environment variables discussed in Chapter 1, the following environment variables must be modified or added to run the Xilinx/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. This is due to the fact that Viewlogic and Xilinx use different versions of Flex/LM licensing that use different delimiters in this variable.

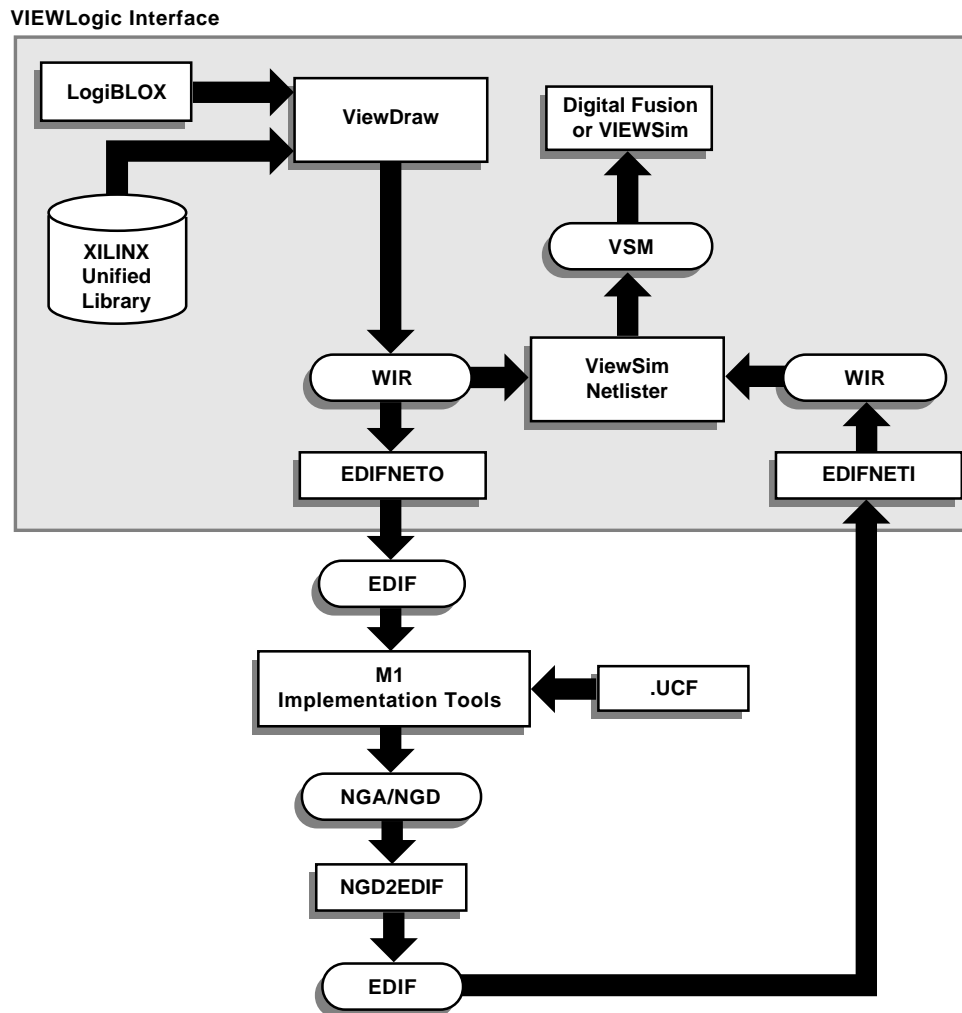
For Windows NT 4.0 users only, select **Start**→**Settings**→**Control Panel**. Double click on the System icon and select the Environment tab. Verify the settings shown above are listed in either the System Variables section or the User Variables section. They will not appear exactly as shown above; the variable will be shown under the Variable header and the path will be shown under the Value header. The word “set” will not appear.

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

Viewlogic/M1 Software Design Flow

Refer to Figure E-1 and the design flow of Viewlogic and the M1 Core Technology tools. The design flow shows design entry, functional simulation, implementation, and timing simulation.

- At the top, 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.
- 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 M1 Core Technology 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.



X8041

Figure E-1 Viewlogic/M1 Core Technology Design Flow

Setting Up Project Libraries

This section describes project library setup for the Workstation, and the PC.

On Workstations

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 Core Technology 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/xc4000ex (xc4000ex)
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)
```

Features of this search order:

- The LogiBLOX library replaces XBLOX. 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 pulldown menus.

1. **Add->LogiBLOX** is used to create a new LogiBLOX component.
2. **Change->LogiBLOX** is used to modify an existing LogiBLOX component.

On PCs

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 Core Technology tools, the M1 Libraries must be used. These libraries must be defined in the Viewlogic project file (.VPJ), located in the project's working directory .

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\xc4000ex (xc4000ex)
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 and Tutorials Guide.

Features of this search order:

- The LogiBLOX library replaces XBLOX. 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 tools pulldown menu. You must initialize these commands by entering an MS-DOS session and executing the following command:

cust menu *xilinx-path\viewlog\data\viewblox.txt*

1. **Tools->Add LogiBLOX** is used to create the new LogiBLOX component.
2. **Tools->Change LogiBLOX** is used to modify an existing LogiBLOX component.

See *Appendix F* for more information on the use of 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 (i.e. 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 (e.g., XC4000EX). 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, please refer to the “Viewlogic Interface/Tutorial Guide”. For more information on timing constraints, see the XACT-Performance Utility chapter of the “Development System Reference Guide”.

Using Special XC4000EX Features

There are new components available for the XC4000EX family. This section will explain how these components are to be added to your schematic so you can take advantage of all the features of this family.

Global Clock Buffers

- BUFGLS — Global Low-Skew Buffers (BUFGLS) are the standard clock buffers. They should be used for most internal clocking whenever a large portion of the device must be driven.
- BUFGE — Global Early Buffers (BUFGE) are designed to provide faster clock access, but CLB access is limited to one-fourth of the device. They also facilitate a faster I/O interface.

- **BUFFCLK** — FastCLK Buffers (BUFFCLK) are specifically designed to provide the fastest possible I/O clock. They have only the standard input access to CLBs, through local interconnect.

IOB Fast Capture Latches

The IOB-Fast Catch Capture Latches are used in the same manner as other input flip-flops. Simply connect an IPAD to the D pin of an ILFFX type component. An input buffer must not be placed between these components and their IPAD.

- ILFFX
- ILFFXI
- ILFLX

Output Multiplexer/2-Input Functions

The output multiplexer/2-input functions provide simple logic to be added in the IOB before the output buffer. These components are placed just like internal primitives, but an OBUF must immediately follow them before connecting to an OPAD. Use the symbol pin labeled “F” for the signal on the critical path.

- OMUX2
- OAND2
- ONAND2
- OOR2
- ONOR2
- OXOR2
- OXNOR2

CLB Latches

The CLB latches are added like any standard internal primitive or macro.

- LD
- LDCE

LogiBLOX

Introduction

LogiBLOX is an on-screen 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: The Xilinx products which support LogiBLOX are: XC3000A, XC3100A, XC4000E, XC4000EX, and XC5200.

Documentation

The following documentation is available for the LogiBLOX program:

- The *LogiBLOX Reference/User Guide* is available on the CDROM supplied with your software and viewable with a DynaText browser.
- The LogiBLOX online help can be accessed from LogiBLOX.
- The *M1 Software Release Notes* describes installation setup and current issues regarding the use of LogiBLOX. *Notes* is available on the supplied CDROM, and viewable with a DynaText browser.
- The *Conversion Guide* compares XBLOX and LogiBLOX, and how to convert an XBLOX design to LogiBLOX. The *Conversion Guide*

and other application notes can be found in the XBBS directory of the Xilinx M1 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 Chapter 1 and in the appropriate appendix in this manual.

Mentor Interface Environment Variables

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

```
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\
```

For 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 Chapter 1 of this 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 two ways:

- From a third-party vendor tool by selecting the menu item that lists LogiBLOX as a menu choice
- From a workstation command line by entering:

```
lbgui
```

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.

Creating a LogiBLOX Module

To create a LogiBLOX module, proceed with the following four steps.

1. In ViewDraw, select the appropriate menu choice in your design tool that will launch the LogiBLOX GUI. The LogiBLOX Module Selector dialog box displays.

- In Mentor Graphics, from Pld_da select:

`Library -> Xilinx Library -> LogiBLOX`

An intermediate dialog window called create/modify/instantiate LogiBLOX symbol appears on-screen before the LogiBLOX GUI displays. This window replaces the LogiBLOX Setup dialog box.

- In Viewlogic, from the ViewDraw window select, for workstation users (on Powerview):

`Add -> LogiBLOX`

- for PC users (on Workview Office):

`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 on OK.

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

Note: For either PC or workstation users, you may add existing LogiBLOX components by taking them 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 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 (4000E and 4000EX), 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 (*.ngo*), 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 “Starting LogiBLOX” in this appendix.
2. Select Setup on the Module Selector dialog box. The Setup dialog box appears on-screen.

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 on-screen.
5. Select the Simulation model you require (VHDL, EDIF, or Verilog).
6. Click on OK. The Setup dialog box will disappear.
7. Create the module you desire in the LogiBLOX Module Selector dialog box.
8. Click on OK.
9. Instantiate the module in the top level.

With a text editor, cut and paste the contents of the VHDL (.vhd) 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/PC command line to analyze the LogiBLOX libraries:

```
qhlib logiblox
qhmap logiblox logiblox
qvhcom -work logiblox
$XILINX/vhdl/src/logiblox/mvutil.vhd
$XILINX/vhdl/src/logiblox/mvlarith.vhd
$XILINX/vhdl/src/logiblox/logiblox.vhd
```

Synopsys VSS

Enter the following series of commands from your workstation/PC 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 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.

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/Subtractor	Clock Divider
Comparator	Constant	Counter
Data Register	Decoder	Input/Output
Memory	Multiplexer	Pad
Shift Register	Simple Gates	Tristate

Appendix G

Instantiated Components

Introduction

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. Associated instantiation can be used to include the component in an HDL design. For complete lists of the Xilinx components, see the CDROM “Libraries Guide” or the “Synopsys (XSI) Interface Tutorial Guide”.

STARTUP Component

The STARTUP component is typically used to access the global set/reset and global 3-state 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 “Programmable Logic Data Book” and the CDROM “Libraries Guide”.

Table G-1 Startup Library Component

Name	Family	Description	Outputs	Inputs
STARTUP	4000E/L, 4000EX, 4000XL, 5200/L ¹	Used to connect Global Set/Reset, global 3-state control, and user configuration clock.	Q2, Q3, Q1Q4, DONEIN	GSR, GTS, CLK

1. 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 CDROM “Libraries Guide”.

Table G-2 BSCAN Library Components

Name	Family	Description	Outputs	Inputs
BSCAN	4000E/L, 4000EX, 4000XL, 5000/L ¹	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, 5000/L	Connects to the BSCAN TDI input. Loads instructions and data on each low-to-high TCK transition.	I	—
TDO	4000E/L, 4000EX, 4000XL, 5000/L	Connects to the BSCAN TDO output. Provides the boundary-scan data on each low-to-high TCK transition.	—	O
TMS	4000E/L, 4000EX, 4000XL, 5000/L	Connects to the BSCAN TMS input. It determines which boundary scan is performed.	I	—
TCK	4000E/L, 4000EX, 4000XL, 5000/L	Connects to the BSCAN TCK input. Shifts the serial data and instructions into and out of the boundary-scan data registers.	I	—

1. 5200 has three additional 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 “Programmable Logic Data Book” and the CDROM “Libraries Guide”.

Table G-3 Readback Library Components

Name	Family	Description	Outputs	Inputs
READBACK	4000E/L, 4000EX, 4000XL, 5200/L	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, 5200/L	Connects to the Mode 0 (M0) input pin, which is used to determine the configuration mode.	I	—
MD1	4000E/L, 4000EX, 4000XL, 5200/L	Connects to the Mode 1 (M1) input pin, which is used to determine the configuration mode.	—	O
MD2	4000E/L, 4000EX, 4000XL, 5200/L	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 CDROM “Libraries Guide”.

Table G-4 RAM and ROM Library Components

Name	Family	Description	Outputs	Inputs
RAM16X1	4000E/L, 4000EX, 4000XL	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	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	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	A 32-word by 1-bit static read-write random access memory.	O	D, A0, A1, A2, A3, A4, WE

Table G-4 RAM and ROM Library Components

Name	Family	Description	Outputs	Inputs
RAM32X1S	4000E/L, 4000EX, 4000XL	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	A 16-word by 1-bit read-only memory component.	O	A3, A2, A1, A0
ROM32X1	4000E/L, 4000EX, 4000XL	A 32-word by 1-bit read-only memory component.	O	A4, A3, A2, A1, A0

Global Buffers

Each XC4000EX device has 20 actual global buffers: 8 BUFGLSs, 8 BUFES, and 4 BUFFCLKs. 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 Core Technology tools to make an appropriate physical buffer allocation. For more information on the components, see the “Programmable Logic Data Book”.

Table G-5 Global Buffers Library Components

Name	Family	Description	Outputs	Inputs
BUFG	3000A, 3100A, 4000E/L, 4000EX, 4000XL, 5200/L	An architecture-independent global buffer, distributes high fan-out clock signals throughout a PLD device.	O	I

Table G-5 Global Buffers Library Components

Name	Family	Description	Outputs	Inputs
BUFGP ¹	4000E/L, 4000EX, 4000XL	A primary global buffer, distributes high fan-out clock, or control signals throughout PLD devices.	O	I
BUFGS ²	4000E/L, 4000EX, 4000XL	A secondary global buffer, distributes high fan-out clock, or control signals throughout a PLD device.	O	I
BUFGLS	4000EX, 4000XL	Global Low-Skew buffer. BUFGLS components can drive all flip-flop clock pins.	O	I
BUFGE	4000EX, 4000XL	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
BUFFCLK	4000EX, 4000XL	Fast clocks. XC4000EX devices have 4 total, 2 each on the left and right sides. BUFFCLK components can drive all IOB clock pins on their left or right half edge.	O	I

1. BUFGP_F for Synopsys

2. BUFGS_F for Synopsys

Fast Output Primitives

One of the features added to the XC4000EX architecture is the fast output MUX. There is one fast output MUX located in each IOB which 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-6 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 IOB	O	F, I0
ONAND2	4000EX, 4000XL	2-input NAND gate that is implemented in the output multiplexer of the XC4000EX IOB	O	F, I0
OOR2	4000EX, 4000XL	2-input OR gate that is implemented in the output multiplexer of the XC4000EX IOB.	O	F, I0
ONOR2	4000EX, 4000XL	2-input NOR gate that is implemented in the output multiplexer of the XC4000EX IOB.	O	F, I0
OXOR2	4000EX, 4000XL	2-input exclusive OR gate that is implemented in the output multiplexer of the XC4000EX IOB.	O	F, I0
OXNOR2	4000EX, 4000XL	2-input exclusive NOR gate that is implemented in the output multiplexer of the XC4000EX IOB.	O	F, I0
OMUX2	4000EX, 4000XL	2 x 1 MUX implemented in the output multiplexer of the XC4000EX 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 CDROM "Synopsys (XSI) Interface/ Tutorial Guide".

Table G-7 IOB Components

Name	Family	Description	Outputs	Inputs
IBUF	3000A, 3100A, 4000E/L, 4000EX, 4000XL, 5200/L	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, 5200/L	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, 5200/L	Single 3-state output buffer with active-low output enable. (3-state High)	O	I,T
IFD	3000A, 3100A, 4000E/L, 4000EX, 4000XL,	Single input D flip-flop.	Q	D, C

Table G-7 IOB Components

Name	Family	Description	Outputs	Inputs
OFD	3000A, 3100A, 4000E/L, 4000EX, 4000XL	Single output D flip-flop.	Q	D, C
OFDT	3000A, 3100A, 4000E/L, 4000EX, 4000XL	Single D flip-flop with active-high 3-state active-low output enable buffers.	O	D, C, T
IFDX	4000E/L, 4000EX, 4000XL	Single input D flip-flop with clock enable.	Q	D0, D1, S0
OFDX	4000E/L, 4000EX, 4000XL	Single output D flip-flop with clock enable	Q	D, C, CE
OFDTX	4000E/L, 4000EX, 4000XL	Single D flip-flop with active-high tristate and active-low output enable buffers.	O	D, C, CE, T
ILD_1	4000E/L, 4000EX, 4000XL	Transparent input data latch with inverted gate. (Transparent High).	Q	D, G

Instantiated Components

Index

C

Cadence Concept/Verilog Interface, A- 1
 board level simulation, A-11
 cds.lib, A-8
 functional simulation, A-9
 global.cmd, A- 7
 HDL direct, A-9
 master.local, A- 8
 NGDBuild functional simulation, A- 10
 pin locking, A-12
 timing constraints, A-12
 timing simulation, A- 11
 translating to Xilinx EDIF, A- 10
constraint files, 3-9
 creating user, 3-10

D

DC2NCF utility, D-8
design, downloading
 creating PROM, 3-19
 in-circuit debugging, 3-19
 re-entrant route, 3-20
design implementation,
 analyzing reports, 3-6
 place and route,
 configure, 3-5
 translate,
 map, 3-4
Design Manager, 1- 5, 3-4
documentation,
 installing, 1-12, 1-17

E

EBTRC environment variable, 1-13
environment variables,
 check setup with PAR, 1-14, 1-18
 for Core Technology software, 1-12, 1-17
 for LogiBLOX, F-2
 for Mentor, C-1
 for Synopsys, D-1
EPIC, 1-5

F

Flow Engine, 1-5
FPGA Express ,B-1
 design entry, B-1
 design flow, B-3
 installation, B-1
 porting code, B-4
 simulation with, B-2
 timing constraints, B-3

G

Guide for Incremental Design Changes, 1-6

H

Hardware Debugger, 1- 6
hardware requirements,
 PCs, 1-15
 workstations, 1-10
hotline, North America, 1-9

HP-UX

setting environment variables, 1-13

I

implementation,

advanced flows, 3-19

exact guide mode,

leveraged guide mode, 3-14

implementation, guiding an, 3-13

installation, 1-8

CAE interface and libraries, 1-12

core technology, 1-12, 1-16

on-line documentation, 1-12, 1-17

PCs, 1-16

CAE interface and libraries, 1- 16

on-line documentation, 1-17

variable settings, 1-17

Workview Office toolset, 1- 16

workstations, 1- 11

variable settings, 1-12

verifying core technology
software, 1-14

verifying DynaText variable
setting, 1-13

instantiated components, G-1

L

licenses, 1-9

file, 1-9

LogiBLOX, 1-4

and Mentor, F-2

and Synopsys, F-2

and Viewlogic, F-2

environment variables, F-2

HDL synthesis design, F-5

modules, F-7

schematic designs, F-4

starting, F-3

M

Mentor Graphics Interface,

environment variables, C-1, C-6

library locations, C-7

pin locking, C-7

timing constraints, C-7

timing simulation, C-5

translating to Xilinx EDIF, C-5

Mentor Graphics interface, C-1

design flow, C-3

and LogiBLOX, F-2

M1 Software,

design flow, 3-8

supported families, 1-3

N

Netlist Support , 1-5

Netlists, supported, 1-4

O

option selection, 3-8

P

PAR, 1-7

verify setup, 1- 18

place and route, multi-pass, 3-21

PROM File Formatter, 1- 7

R

RAM and ROM components, G-4

READBACK component, G-3

Re-Entrant Routing, 1-7

reports

delay report, 2- 16

map report, 2-10, 3-6

pad report, 2-16, 3-7

place & route report, 2-17, 3-7

summary timing, 3-16

translation report, 2- 10, 3-6

S

- SCALD methodology, A- 9
- schematic designs
 - using LogiBLOX, F-4
- SIZE property, A- 9
- simulation files,
 - creating, 3-17
- Solaris
 - setting environment variables, 1-13
- SunOS
 - setting environment variables, 1-13
- Synopsys,D-1
 - code comments, D-12
 - DC2NCF utility, D-8
 - design flow, D-2
 - documentation available, D-1
 - entity coding, D-9
 - environment variables, D-1
 - FPGA compiler users, D-9
 - script file, D-5
 - setup files, D-3
 - timing constraints, D-8

T

- third party synthesis, 1-8
- Timing Analyzer, 1-7
- timing analysis,
 - after map, 3-15
 - after place and route, 3-15
 - detailed, 3-16
 - static, 3-14
- timing specification performance, 1-8

V

- verify setup, 1-18
- Viewlogic,
 - and LogiBLOX, F-2
 - assigning pin location, E-8
 - CONFIG symbol, E-9
 - design flow, E-4
 - global clock buffers, E-9

- project libraries, E-6

- environment variables, E-1
- libraries, E-4
- on workstations , E-1

W

- www.Zilinx.com, 1-4