

# *Quick Start Guide for Xilinx Alliance Series vM1.3*

*Introduction*

*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*

*M1 Constraints Guide*



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

XILINX, XACT, XC2064, XC3090, XC4005, XC5210, 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, LogiBLOX, 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 Omaton 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. FLEXIm is a trademark of Globetrotter, Inc. DynaText is a registered trademark of Inso Corporation.

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 1991-1997 Xilinx, Inc. All Rights Reserved.

## *Glossary*

# Preface

---

## About This Manual

This quick start guide does what its title purports, provides an overview of the features and additions to Xilinx's newest product: vM1.3 Software. The primary focus of this guide is the Core Technology tools used to implement a design.

## Manual Contents

This manual covers the following topics.

- Chapter 1, "Introduction," introduces the new and enhanced features of the M1 software.
- Chapter 2, "Installation," gives instructions on the installation of the latest Xilinx software on workstations, and PCs.
- Chapter 3, "Core Technology Tutorial," provides a tutorial which exercises a significant portion of the features of the M1 design flow.
- Chapter 4, "How This Release Works," looks in-depth at the capability and flexibility of the Xilinx software.
- Appendix A, "Cadence Concept and Verilog Interface Notes," covers how to set up the Cadence Concept interface for schematic entry, and Verilog-XL for simulation.
- Appendix B, "FPGA Express Interface Notes," covers how to install and start using FPGA Express and the XACTstep vM1.3, Synopsis, and the Xilinx CDRM documentation.
- Appendix C, "Mentor Graphics Interface Notes," covers how to set up the Mentor Graphics interface and associated libraries.

- Appendix D, “Synopsys Interface Notes,” covers how to set up the Synopsys interface and associated libraries.
- Appendix E, “Viewlogic Interface Notes,” covers how to set up the Viewlogic interface and project libraries.
- Appendix F, “LogiBLOX,” covers how to set up the LogiBLOX interface and associated libraries
- Appendix G, “Instantiated Components,” includes a listing of the components most frequently instantiated in synthesis designs.
- Appendix H, “M1 Constraints Guide,” describes the most common constraints you can apply to your design to control the timing and layout of a Xilinx FPGA or CPLD.
- Appendix I, “Glossary,” contains definitions and explanations for terms used in the Quick Start Guide.

There are eight appendices, five of them devoted each to an interface, one to LogiBLOX (including HDL), one to frequently instantiated components, and the last features the M1 constraints guide.

# 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;`

In addition, Xilinx has created several conventions for use within the DynaText online documents.

- Red-underlined text indicates an interbook link, which is a cross-reference to another book. Click on the red-underlined text to open the specified cross-reference.
- Blue-underlined text indicates an intrabook link, which is a cross-reference within a book. Click on the blue-underlined text to open the specified cross-reference.
- There are several types of icons.

Iconized figures are identified by the figure icon.

**Figure 1-1 Naming Conventions**



Iconized tables are identified by the table icon.

**Table 13-14 Carry Modes**



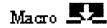


---

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



The DynaText footnote icon displays next to the footnoted text.



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

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



# Contents

---

## Chapter 1 Introduction

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
Flow Engine .....	1-3
LogiBLOX .....	1-3
Timing Specification Performance .....	1-4
Timing Analyzer .....	1-4
Multi-Pass PAR .....	1-4
Re-Entrant Routing .....	1-5
Guide for Incremental Design Changes .....	1-5
EPIC - Physical Design Editor .....	1-5
Hardware Debugger .....	1-6
PROM File Formatter .....	1-6
Third Party Interfaces .....	1-6

## Chapter 2 Installation

Introduction .....	2-1
Technical Support .....	2-2
Hotline Access .....	2-2
Obtaining and Setting Up Licenses .....	2-2
Customer Service .....	2-4
United States and Canada .....	2-4
Europe .....	2-4
Other International Countries .....	2-4
Registering and Licensing from the Web .....	2-4

To generate a license online, the following information is required:.....	2-5
9-Digit End User ID Number (example, 1234-01-01-A) .....	2-5
Product Serial Number .....	2-5
Product Type .....	2-5
Xilinx Web Licensing and Registration Program .....	2-5
Registering Via Telephone/Fax.....	2-5
PC or Workstation.....	2-6
1. What type of machine will be used as the license server: PC or Workstation?.....	2-6
2. What type of machine will actually be running the M1 Core Technology software: PC or Workstation or both? ...	2-6
3. What type of license is needed: node-locked or floating? .....	2-7
Setting Up Your License File .....	2-7
Workstation Users: .....	2-7
PC Users: .....	2-7
M1 Requirements for Workstations.....	2-7
M1 Installation on Workstations .....	2-8
Installing the Core Technology Software .....	2-9
Installing the CAE Interface and Libraries.....	2-9
Installing the On-line Documentation .....	2-10
Variable Settings for Workstations.....	2-10
Verifying Core Technology Software Installation - Workstations .....	2-11
Example 1:.....	2-11
Example 2:.....	2-11
Example 3:.....	2-11
Verifying DynaText Variable Settings - Workstation .....	2-12
Setting Up LogiBLOX for Workstations.....	2-13
M1 Requirements for PCs.....	2-13
M1 Installation on PC.....	2-13
Installing Core Technology Software, CAE Interface and Libraries .....	2-14
Installing Workview Office Toolset .....	2-14
Installing On-line Documentation .....	2-15
Variable Settings for PCs.....	2-15
Setting Up LogiBLOX for Use With Workview Office .....	2-16

## **Chapter 3 Core Technology Tutorial**

Tutorial Installation .....	3-1
-----------------------------	-----

Step 1: Invoking Design Manager, Creating an Implementation Project .....	3-2
Step 2: Creating Design Versions, Implementation Revisions .....	3-4
Step 3: Mapping a Design .....	3-6
Step 4: Using Timing Analyzer to Evaluate Block Delays After Mapping .....	3-12
Step 5: How to Place and Route a Design .....	3-14
Step 6: Evaluating With Worst Case Timing .....	3-18
Step 7: Using the Flow Engine to Create Timing Simulation Data .....	3-20
Step 8: Using the Flow Engine to Create Configuration Data .....	3-22
Step 9: Using the PROM File Formatter to Create PROM Files ...	3-24

## Chapter 4 How This Release Works

Starting Xilinx Tools .....	4-11
Creating A Project .....	4-13
Implementing A Design .....	4-14
Translate .....	4-14
Map .....	4-15
Place and Route .....	4-15
Configure .....	4-16
Analyzing Reports .....	4-16
Translation Report .....	4-16
Map Report .....	4-16
Place and Route Report .....	4-17
Pad Report .....	4-17
Selecting Options .....	4-18
Using Constraint Files .....	4-110
Design, Netlist, User, and Physical Constraints .....	4-110
Creating A User Constraint File .....	4-110
Guiding An Implementation .....	4-114
Exact Guide Mode .....	4-115
Leveraged Guide Mode .....	4-115
Static Timing Analysis .....	4-115
Static Timing Analysis After Map .....	4-115
Static Timing Analysis After Place and Route .....	4-116
Summary Timing Reports .....	4-116
Detailed Timing Analysis .....	4-117
Creating Simulation Files .....	4-117
When Can Simulation Data Be Created .....	4-118
Creating Timing Simulation Data .....	4-118
Creating Functional Simulation Data .....	4-119
Downloading A Design .....	4-119

Creating a PROM.....	4-120
In-Circuit Debugging .....	4-120
Advanced Implementation Flows .....	4-120
Re-Entrant Route .....	4-120
Multi-Pass Place and Route.....	4-122

## **Appendix A Cadence Concept and Verilog Interface Notes**

Documentation .....	A-1
Setting up the Xilinx/Cadence Interface .....	A-2
Cadence/Verilog and M1 Design Flow.....	A-4
Setting Up for Concept.....	A-8
Using HDL Direct .....	A-10
Iterated Instances Vs. Size Support.....	A-10
Starting Up Concept.....	A-10
Functional Simulation.....	A-11
Testfixture: Asserting the Global Set Reset in a Pre-NGDBuild	
Unified Library Functional Simulation .....	A-11
Pure Concept Schematic Functional Simulation.....	A-13
Post-NGDBuild Functional Simulation .....	A-13
Translating a Design to Xilinx EDIF .....	A-14
Timing Simulation.....	A-14
Support for Board Level Simulation .....	A-15
Pin Locking.....	A-15
Timing Constraints .....	A-16

## **Appendix B FPGA Express Interface Notes**

Installation of FPGA Express .....	B-1
Design Entry With FPGA Express.....	B-2
Simulation With FPGA Express .....	B-2
Documentation .....	B-3
FPGA Express/XC4000EX Pre-Release Design Flow.....	B-3
Timing Constraints .....	B-3
Porting Code from FPGA Compiler to FPGA Express.....	B-4

## **Appendix C Mentor Graphics Interface Notes**

Documentation .....	C-1
Setting up the Xilinx/Mentor Interface .....	C-2
Mentor/M1 Software Design Flow .....	C-3
Translating a Design to Xilinx EDIF .....	C-6
Timing Simulation.....	C-6
Generating a Timing-Annotated EDIF Netlist.....	C-6
Generating a Timing Model.....	C-6

Creating a Simulation Viewpoint .....	C-7
Running PLD_QuickSim .....	C-7
Mentor-Related Environment Variables .....	C-7
Library Locations and Sample MGC Location Map.....	C-8
Pin Locking.....	C-8
Timing Constraints .....	C-8

## Appendix D Synopsys Interface Notes

Documentation .....	D-1
Setting Up the Xilinx/Synopsys Interface .....	D-2
Synopsys/M1 Software Design Flow.....	D-3
Examples of Synopsys Setup Files .....	D-3
.synopsys_dc.setup .....	D-3
.synopsys_vss.setup .....	D-6
Example Script File .....	D-7
Timing Constraints and DC2NCF.....	D-9
DC2NCF Design Flow.....	D-10
FPGA Compiler Users.....	D-10
Entity Coding Examples .....	D-10
VHDL Code.....	D-10
Verilog Code: Module Example .....	D-13
Comments About Code.....	D-14

## Appendix E Viewlogic Interface Notes

Documentation .....	E-1
Setting Up Viewlogic Interface on Workstations .....	E-1
Setting Up Xilinx/Viewlogic Interface on the PC.....	E-3
Viewlogic/M1 Software Design Flow .....	E-4
Setting Up Project Libraries .....	E-6
On Workstations .....	E-6
Xilinx Commands in ViewDraw .....	E-7
On PCs .....	E-7
Xilinx Commands in ViewDraw .....	E-8
Assigning a Pin Location.....	E-8
Timing Constraints.....	E-9
Using Special XC4000EX Features .....	E-9
Global Clock Buffers .....	E-9
IOB Fast Capture Latches .....	E-10
Output Multiplexer/2-Input Functions .....	E-10
CLB Latches .....	E-10

## Appendix F LogiBLOX

Documentation .....	F-1
Setting Up LogiBLOX on a Workstation .....	F-2
Mentor Interface Environment Variables .....	F-2
Synopsys Interface Environment Variables .....	F-2
Viewlogic Interface Environment Variables .....	F-3
Setting Up LogiBLOX on a PC .....	F-3
Viewlogic Interface Environment Variables .....	F-3
Starting LogiBLOX .....	F-4
Using LogiBLOX for Schematic Design .....	F-4
Creating a LogiBLOX Module .....	F-4
Design simulation .....	F-5
Copying Modules .....	F-5
Using LogiBLOX for HDL Synthesis Design .....	F-6
Instantiating a LogiBLOX Module .....	F-6
Analyzing a LogiBLOX Module .....	F-6
Mentor QuickHDL .....	F-7
Synopsys VSS .....	F-7
Viewlogic Vantage .....	F-7
LogiBLOX Modules .....	F-8

## **Appendix G Instantiated Components**

STARTUP Component .....	G-1
BSCAN Component .....	G-2
READBACK Component .....	G-3
RAM and ROM .....	G-4
Global Buffers .....	G-5
Fast Output Primitives .....	G-7
IOB Components .....	G-8

## **Appendix H M1 Constraints Guide**

Constraint Entry Mechanisms .....	H-1
Translating and Merging Logical Designs .....	H-3
Constraints File Overview .....	H-4
The Netlist Constraint File (NCF) .....	H-4
The User Constraint File (UCF) .....	H-5
The Physical Constraints File (PCF) .....	H-5
Case Sensitivity .....	H-6
UCF Timing Constraints .....	H-6
The “From:To” Style Timespec .....	H-6
Using TPSYNC .....	H-8
The Period Style Timespec .....	H-9
The Offset Constraint .....	H-10



Ignoring Paths.....	H-11
Controlling Skew .....	H-12
Constraint Precedence .....	H-12
Layout Constraints .....	H-14
Converting a Logical Design to a Physical Design .....	H-14
Last One Wins Resolution .....	H-15
XC5200 Constraints.....	H-15
Efficient Use of Timespecs and Layout Constraints.....	H-16
The “Starter Set” of Timing Constraints .....	H-16
Standard Block Delay Symbols.....	H-18
Table of M1-Supported Constraints .....	H-20
Basic UCF Syntax Examples .....	H-22
PERIOD TIMESPEC.....	H-22
FROM:TO TIMESPECs .....	H-22
OFFSET TIMESPEC .....	H-23
TIMING IGNORE .....	H-23
PATH EXCEPTIONS .....	H-24
MISCELLANEOUS EXAMPLES .....	H-24
Constraining LogiBLOX RAM/ROM with Synopsys .....	H-25
Referencing a LogiBLOX Module/Component in the FPGA/Design Compiler and FPGA Express Flow .....	H-26
Referencing the Primitives of a LogiBLOX Module in the FPGA/Design Compiler and FPGA Express Flow .....	H-27
FPGA/Design Compiler and Express Verilog Example .....	H-28
test.v: .....	H-28
inside.v:.....	H-28
memory.v (FPGA/Design compiler only) .....	H-29
runscript (FPGA/Design compiler only) .....	H-29
test.ucf (FPGA/Design compiler only).....	H-29
test.ucf (FPGA Express only) .....	H-29
FPGA/Design Compiler and Express VHDL Example .....	H-30
test.vhd .....	H-30
inside.vhd.....	H-31
runscript (FPGA/Design compiler only) .....	H-31
test.ucf (FPGA/Design compiler only).....	H-32
test.ucf (FPGA Express only) .....	H-32

## Appendix I Glossary

Definitions .....	I-1
aliases.....	I-1
attribute .....	I-1
AutoRoute .....	I-1

block.....	I-1
component .....	I-1
constraint .....	I-2
Core Technology Tools.....	I-2
DC2NCF .....	I-2
guided mapping .....	I-2
HDL.....	I-2
LCA file .....	I-2
LCA2NCD .....	I-3
LogiBLOX.....	I-3
locking.....	I-3
logic.....	I-3
Logic Block Editor .....	I-3
macro .....	I-3
MCS file .....	I-4
MDF file.....	I-4
MRP file .....	I-4
NCD file.....	I-4
NCF file.....	I-4
NGDAnno.....	I-5
NGA file.....	I-5
NGD2EDIF .....	I-5
NGD2XNF .....	I-5
NGD2VER.....	I-5
NGD2VHDL .....	I-5
NGDBuild .....	I-5
NGD file .....	I-6
NGM file .....	I-6
PAR (Place and Route).....	I-6
path delay .....	I-6
PCF file .....	I-6
physical Design Rule Check (DRC) .....	I-6
physical macro .....	I-7
pin .....	I-7
pinwires.....	I-7
route.....	I-7
route-through .....	I-7
states .....	I-7

TRCE .....	1-8
TWR file .....	1-8
wire .....	1-8
UCF file .....	1-8



# Chapter 1

## Introduction

---

Version M1.3 software will henceforth be referred to as M1 Software. This chapter contains the following sections.

- “Introduction” section
- “M1 Software Supported Families” section
- “Supported Netlists” section
- “M1 Software Instruction Volumes” section
- “New and Enhanced Tools” section
- “Third Party Interfaces” section

## Introduction

Welcome to Xilinx’s newest software release, vM1.3, 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:** 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 Series F2.1 tools.

## M1 Software Supported Families

M1 Software supports the following families with earlier version compatibility: 4000E, 4000L, 7300, and 9500. 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 is included in the Xilinx “Programmable Logic Data Book”.

## Supported Netlists

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

Netlists	Variations
EDIF	Multiple variations are accepted, including: SEDIF EDIF
XNF	Multiple variations are accepted, including: SXNF XFF

**Note:** All designs must be created using the Xilinx Unified Libraries. A list of components is 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
- Flow Engine
- LogiBLOX
- Netlist Support
- Timing Specification Performance
- Timing Analyzer
- Multi-Pass PAR
- Re-Entrant Routing

- Guide for Incremental Design Changes
- EPIC-the Physical Design Editor
- Hardware Debugger
- PROM File Formatter

## 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.

## 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

## LogiBLOX

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

- 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 and some schematic capture programs, or can be integrated with Viewlogic or Mentor Graphics schematic capture software. Refer to the “LogiBLOX” appendix for further details.

## Timing Specification Performance

Xilinx M1 Software 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

## 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 design directly out of MAP, or on the block and route delays of a design already placed and routed.

## 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 CPU time, instead of your time, to achieve faster design results.

## 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.

## 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 re-synthesis of your design.

## 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:

- Ability to view 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

## Hardware Debugger

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.

## 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.

## Third Party Interfaces

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

## Installation

---

This chapter contains the following sections.

- “Introduction” section
- “Obtaining and Setting Up Licenses” section
- “Customer Service” section
- “Registering and Licensing from the Web” section
- “Xilinx Web Licensing and Registration Program” section
- “Registering Via Telephone / Fax” section
- “M1 Requirements for Workstations” section
- “M1 Installation on Workstations” section
- “M1 Requirements for PCs” section
- “M1 Installation on PC” section

## Introduction

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.

## Technical Support

### Hotline Access

Location	Telephone	Electronic Mail
U.S. and Canada	1-800-255-7778	hotline@xilinx.com
Japan	81-33-297-9163	jhotline@xilinx.com
France	33-1-3463-0100	frhelp@xilinx.com
Germany	49-89-9915-4930	dlhelp@xilinx.com
United Kingdom	44-1-932-820821	ukhelp@xilinx.com
To Contact Factory	1-408-879-5199	hotline@xilinx.com


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. To obtain a license, you need to be a registered user in the Customer Service database. New Xilinx users should fill out their Xilinx registration card and **fax or mail** it to their Customer Service location or can register online. Customer Service will send your license and authorization codes in a *license.dat* file.

You can obtain a license by accessing the online Web tool or by contacting your local Xilinx Customer Service representative. If you request your license via fax, please fill out the "M1 License Request Form" figure 2-1 in this chapter.

**Note:** This form is also located in the front cover of your hardcopy version.

<b>M1 License Request Form</b>			
<i>Please complete and fax this form to your local Xilinx Customer Service representative.</i>			
			_____ Xilinx End User Code
_____ End User Name			
_____ Company			
_____ Shipping Address (Mailstop) Do not use P.O. Box			
_____ State/Province	_____ Zip/Postal Code	_____ Country	
_____ Telephone	_____ FAX	_____ E-Mail	
_____ Xilinx Product Part Number		_____ Serial/Key Number	
_____ Ethernet Address/C-Drive Serial Number or Host ID & Server Name for Workstation			
Note: This form is also located in the bookshelf box containing your software and guides.			

X8089

**Figure 2-1 M1 License Request Form**

## Customer Service

Information for contacting your local Xilinx Customer Service representative.

### United States and Canada

Monday-Friday, 8:00am to 5:00pm Pacific Standard Time  
1-800-624-4782 and facsimile 408-559-0115.

### Europe

Monday-Friday, 9:00am to 5:30 pm United Kingdom time - English speaking only.

Country	Telephone	Facsimile
United Kingdom	01932-333550	01932-828521
Belgium	0800 73738	
France	0800 918333	
Germany	0130 816027	
Italy	1677 90403	
Netherlands	0800 0221079	
Other European Locations	(44) 1932-333550	(44) 1932-828521

### Other International Countries

Japan	81 3 3297 9153	81 3 3297 9189
Southeast Asia/ROW	Contact Local Distributor	Contact Local Distributor

International countries not listed, please contact your local distributor.

## Registering and Licensing from the Web

Existing customers can interact with the online license tool and execute and receive a license with immediate turnaround. New customers can register online and receive a license for their product within 24 hours from their customer service representative.

## To generate a license online, the following information is required:

### **9-Digit End User ID Number (example, 1234-01-01-A)**

For existing customers, this is located on the shipping box label. New customers will receive their end user ID after their product has been registered.

### **Product Serial Number**

This is located on the shipping box label and /or registration card. Update products do not include a registration card or a serial number.

### **Product Type**

Example, DS-ALI-STD-PC.

If you do not have the above information, please contact your Customer Service representative or local distributor.

## **Xilinx Web Licensing and Registration Program**

You now have the ability to license and register your software on the Web. Use the following instructions to locate Xilinx licensing and registration program on the Worldwide Web.

Internet address: <http://www.xilinx.com/support/support.htm>

- Go to Xilinx home page (<http://www.xilinx.com>)
- Click on the **Support** hyperlink
- Click on the **Software Licensing and Registration** hyperlink

## **Registering Via Telephone/Fax**

Please fill out, photocopy, and fax the "M1 License Request Form" in Figure 2-1 to your local Customer Service representative in the United States and Canada (408) 559-0115. European customers may provide required information via email at [m1license@xilinx.com](mailto:m1license@xilinx.com), or by fax (44) 1932-828521. International customers may also contact their local distributor for license information.

## PC or Workstation

When you telephone / fax your local Xilinx Customer Service representative for your license, please be prepared to answer the following questions:

### 1. What type of machine will be used as the license server: PC or Workstation?

If the answer is PC, you will need to provide one of the following.

- The Ethernet Address of the PC the M1 license server will be running on. Open an MS-DOS session and type `<path_to_xilinx>\bin\nt\lmutil lmhostid`, which will return the 12 digit hexadecimal address. The `path_to_Xilinx` can be to an installed location or to the Core Technology CD.

If you do not have an Ethernet Address the following is a second option.

- The 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 type `VOL C:`, which will return the 8 digit hexadecimal serial number. Please provide the C:\ drive serial number even if the M1 Core Technology tools are to installed to a different drive letter.

The Ethernet address is preferred over the C:\ drive Serial Number due to its relatively greater stability.

If the answer is Workstation, you will need to provide the hostname and hostid of the server where the license server will be running.

### 2. What type of machine will actually be running the M1 Core Technology software: PC or Workstation or both?

The software may be run on the same machine as the license server, or may be run on another machine within the same network as the machine running the license server. They do not have to be the same platform. Simply assign the `LM_LICENSE_FILE` variable to point to the `license.dat`, and make sure that the license manager is running on the server machine.



### 3. What type of license is needed: node-locked or floating?

A node-locked license allows the M1 Core Technology software to be run only on the local machine. A floating license allows products to be checked out to be run on the local machine, or to be run on a machine on the network. A floating license may contain multiple license for each product.

## 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:\flexlm 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 7.31 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 "Setting Up Security" section of the *Release Document*.

## 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, 2.4 and 2.5

- Ultra Sparc (or equivalent)
- HP-UX HP 10.1 and 10.2
- HP715 (or equivalent)

**Table 2-1 Memory Requirements**

<b>Xilinx Device</b>	<b>RAM</b>	<b>Swap Space</b>
XC4000E/L XC4028EX through XC4036EX XC4005XL through XC4028XL XC7300 XC9500/F (small devices)	64 MB	64 MB-128 MB
XC4036XL through XC4062XL XC9500/F (large devices)	128 MB	128 MB-256 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

For more detailed information about Workstation Installation, refer to the “Workstation Installation” chapter of the *Release Documentation*.

All workstation installations must be done while logged in with “root” authority.

	<b>Required Installation, Verification Checklist</b>
	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)
<b>1.</b>	Obtain license and authorization code for each product
<b>2.</b>	Install Core Technology software
<b>3.</b>	Install CAE interface and Libraries
<b>4.</b>	Install On-line Documentation
<b>5.</b>	Setup license file
<b>6.</b>	Set environment variables
<b>7.</b>	Verify Dyna Text and variable settings
<b>8.</b>	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 Document*.

## Installing the CAE Interface and Libraries

Installation of the CAE interface and libraries 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

**Note:** If you have previously installed the Beta/Pre-release version of the M1 software you must re-load the Dyna Text Browser Software.

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
- 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 LM_LICENSE_FILE $XILINX/data/license.dat
```

For SunOS and Solaris only:

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

For HP-UX only:

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

For example (SunOS):

```
setenv XILINX /usr/xilinx
set path = ($XILINX/bin/sun $path)
```

```
setenv LM_LICENSE_FILE $XILINX/data/license.dat
setenv LD_LIBRARY_PATH $XILINX/bin/sun:/usr/
openwin/lib
```

## 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.

## Verifying DynaText Variable Settings - Workstation

The M1 Documentation is located in the `$XILINX/data/dtext` directory. The documentation must be viewed using the DynaText browser supplied on the CDROM and installed during the Core Technologies software installation. The browser is installed in the `$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:

```
dtext
```

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 "Workstation Installation" chapter of the *Release Document*.

## Setting Up LogiBLOX for Workstations

Refer to “LogiBLOX” appendix of this manual, and the “Setting Up LogiBLOX on a Workstation” section.

### M1 Requirements for PCs

The M1 Software supports the following PC operating systems:  
Windows 95 and Windows NT 4.0 .

**Table 2-2 Memory Requirements for PCs**

Xilinx Device	RAM	Virtual Memory
XC4003E/L through XC4008E/L XC4005XL through XC4008XL XC7300 XC9500/F (small devices only)	32 MB	32 MB-64 MB
XC4010E/L through XC4025E/L XC4028EX through XC4036EX XC4010XL through XC4028XL XC9500/F (medium devices only)	64 MB	64 MB-128 MB
XC4036XL through XC4062XL XC9500/F (large devices)	128 MB	128 MB -256 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

For more detailed information about PC Installation, refer to the “PC Installation” chapter of the *Release Document*.

<b>Required Installation, Verification Checklist</b>	
	For Windows 95, and Windows NT 4.0
<b>1.</b>	Obtain license and authorization code for each product
<b>2.</b>	Install Core Technology software, CAE interface and Libraries
<b>3.</b>	Install Workview Office 7.31 (only if using Viewlogic)
<b>4.</b>	Install On-line Documentation
<b>5.</b>	Setup license file
<b>6.</b>	Set environment variables
<b>7.</b>	Setup LogiBLOX for use with ViewDraw

## **Installing Core Technology Software, 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 three steps.

1. Insert the CDROM labeled "Workview Office 7.31" in the CDROM drive.
2. Run *wvoinst.exe* located in the CDROM root directory.
3. Consult the Viewlogic Interface and Tutorial Guide for more information on the Workview Office installation.



## 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:\flexlm\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:\flexlm\license.dat
set WDIR=c:\wvoffice\standard
set VANTAGE_VSS=c:\wvoffice\v
set VANTAGE_CC=c:\wvoffice\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

The 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 a PC” section of Appendix F.

## Core Technology Tutorial

---

This chapter contains the following sections.

- “Tutorial Installation”
- “Step 1: Invoking Design Manager, Creating an Implementation Project”
- “Step 2: Creating Design Versions, Implementation Revisions”
- “Step 3: Mapping a Design”
- “Step 4: Using Timing Analyzer to Evaluate Block Delays After Mapping”
- “Step 5: How to Place and Route a Design”
- “Step 6: Evaluating With Worst Case Timing”
- “Step 7: Using the Flow Engine to Create Timing Simulation Data”
- “Step 8: Using the Flow Engine to Create Configuration Data”
- “Step 9: Using the PROM File Formatter to Create PROM Files”

### Tutorial Installation

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:

<code>count8.sxnf</code>	<b>Synopsys Xilinx Netlist file</b>
<code>count8.ncf</code>	<b>Netlist Constraints File</b>
<code>count8.ucf</code>	<b>User Constraints File</b>
<code>timing.pcf</code>	<b>Physical Constraints File</b>

## 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 selecting **Start** → **Programs** → **Xilinx** → **Design Manager**.

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.

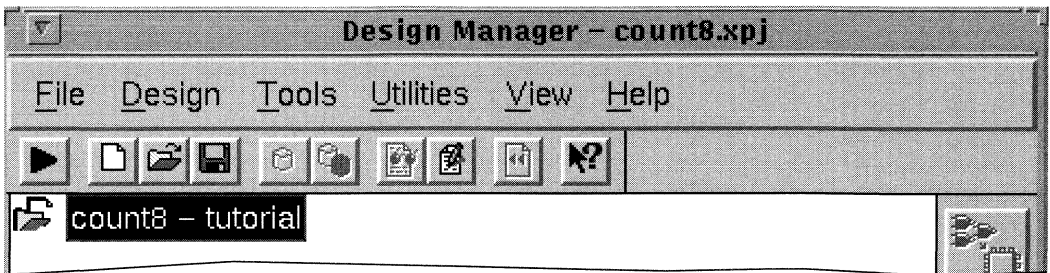
The Browse dialog is displayed with the default file type set to EDIF Files. The design of this tutorial was created by Synopsys FPGA Compiler. Therefore, open an SXNF file.

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.

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

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. Since we will use this we 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 the "Project "count8 - tutorial" Begun" figure.



**Figure 3-1 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 implementation data stored in the Xilinx implementation project area. Because you might try different implementation strategies, you may also have several revisions for a single version.

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

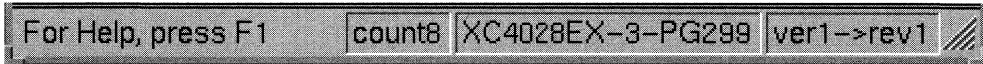
1. **Select Design** → **New Version** to create the first design version.  
The New Version dialog is displayed with the default version being 'ver1'.
2. Click OK to create the new version.  
The Design Manager now shows 'ver1' under the count8 design project.
3. **Select Design** → **New Revision** 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. 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.

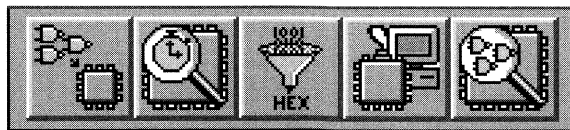
At the bottom of the Design Manager is the status bar (refer to the "Design Manager Status Bar" figure). The status bar contains information such as the current project, target device, and currently selected version → revision pair. The leftmost portion of the status bar (not shown in the figure) is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



**Figure 3-2 Design Manager Status Bar**

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

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



**Figure 3-3 Toolbox shown in horizontal placement**

## Step 3: Mapping a Design

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

The first stage is to translate the input netlist and merge it into a single design file. The design is then mapped into CLBs and IOBs. The mapped design then gets placed and routed. Finally, the configure step creates a configuration bitstream which can then be downloaded to the target system or formatted into a PROM programming file by the PROM formatter.

The Flow Engine gives you, the designer, complete control over 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 4 of this manual.

2. Select **Setup** → **Options** to open the implementation Options dialog.

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

3. 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.

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



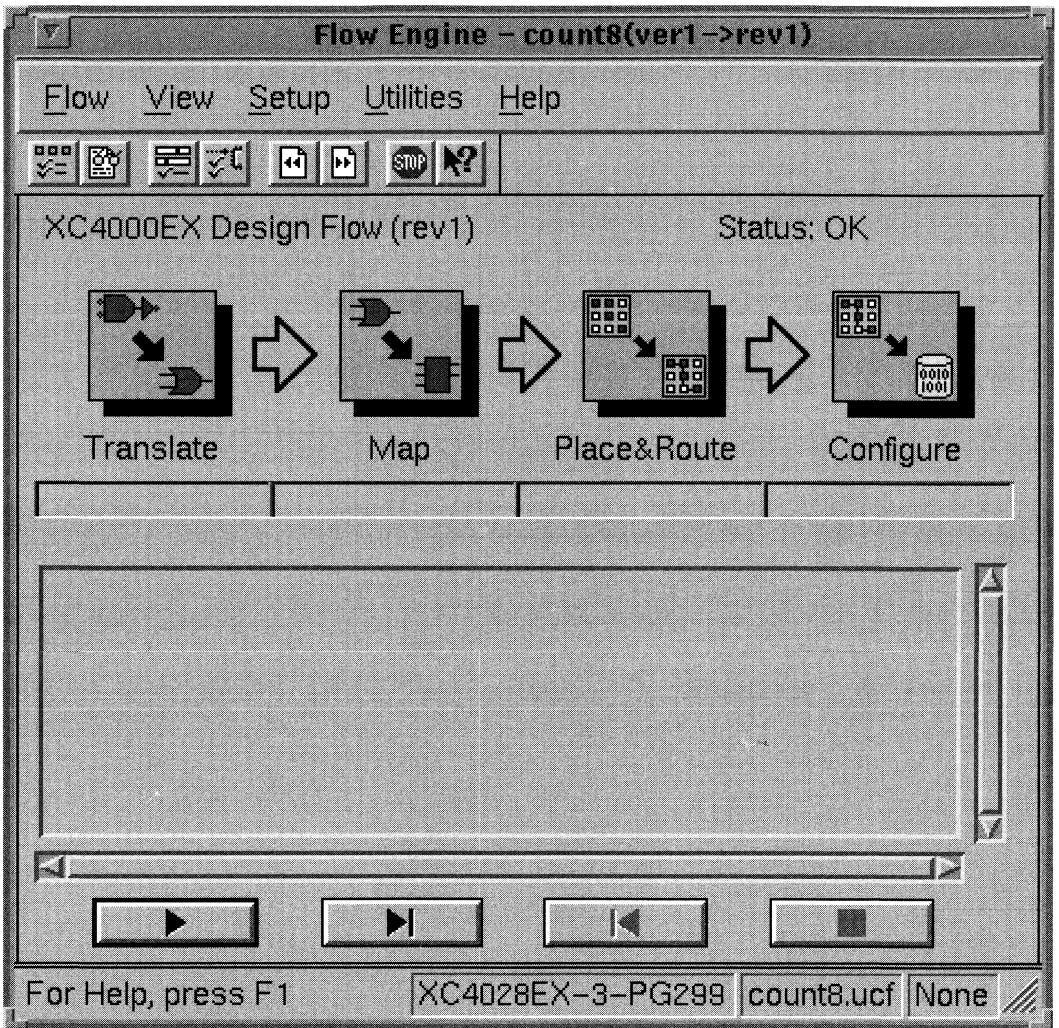
The User Constraints field of the Options dialog is updated with the specified constraints file. At this point we want to map the design's logic and, therefore, do not need to modify the templates or specify any additional targets.

5. Click OK to close the Options dialog.

Refer to the "Flow Engine With "count8.ucf" Selected" figure. 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 stop after translating and mapping the design. Therefore, specify that the Flow Engine stop after the map process by setting a break point.

6. Click on the stop sign toolbar icon (refer to the "Flow Engine With "count8.ucf" Selected" figure).

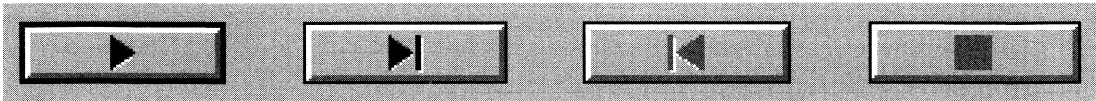


**Figure 3-4 Flow Engine With “count8.ucf” Selected**

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.

7. 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” allows 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 3-5, can be selected.



**Figure 3-5 Control Panel, left to right, Play, Step Forward, Step Backward, and Stop (Press <F1> for on-line Help).**

8. Click on the 'run' control button (far left on the Control Panel, Figure 3-5) 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 following chapters in the *Development System Reference Guide*.

- “NGD2EDIF” chapter
- “NGD2XNF” chapter
- “NGD2VER” chapter
- “NGD2VHDL” chapter

Map is the next program run by the Flow Engine (refer to the “Flow Engine Shows Completed Map Stage” figure).

- 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.

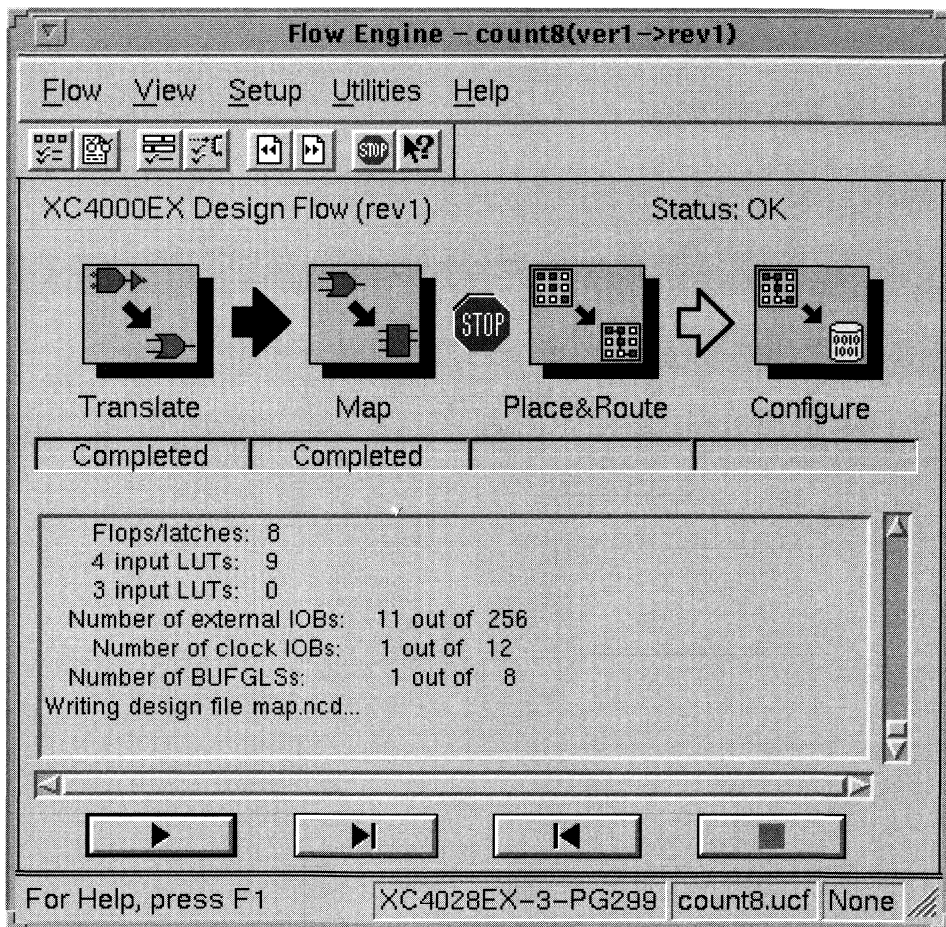
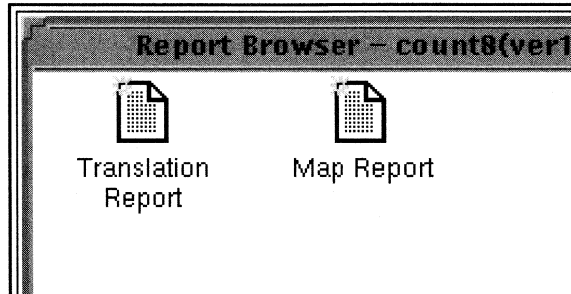


Figure 3-6 Flow Engine Shows Completed Map Stage

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

9. 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 3-7 Report Browser shows reports available after map stage.**

10. Double-click on the Map Report to review the Map process output.
  - **Map Report** contains information on how the target device resources were allocated, 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).

11. 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 net delays are estimated.

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 path containing 10ns of block delay 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 to 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.

4. Select the `timing.pcf` file and click OK or Open 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 timing analysis, the timing constraint will be viewed as impossible to meet. If the total block delay exceeds the timing specification for example, the total block delay of a path is 30ns and the timing specification covering that path is 25ns, the spec is viewed as impossible. 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 16ns after routing the design, given that the block levels currently contribute about 8ns. The net delays listed are actually estimates of what might happen once the design is placed and routed.

**Note:** Regarding the status of the speeds files, the actual numbers displayed in your report may differ.

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.

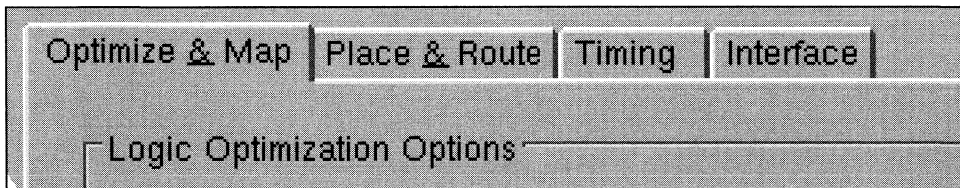
The Flow Engine is invoked in the mapped state. First we must specify the desired options to run the place and route stage.

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 the “Implementations Options” figure).



**Figure 3-8 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.

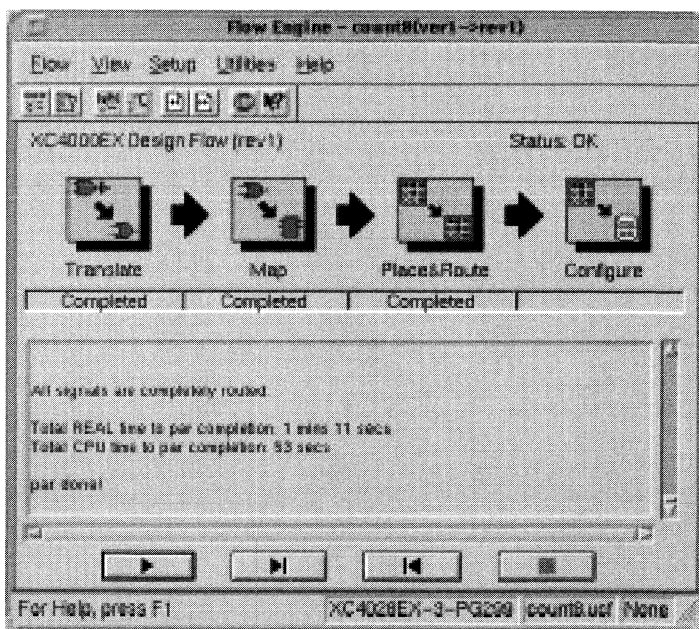
5. While holding the left mouse button on the placer slider, slide the control completely to the left.

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 the “Place and Route Stage Completed” figure.

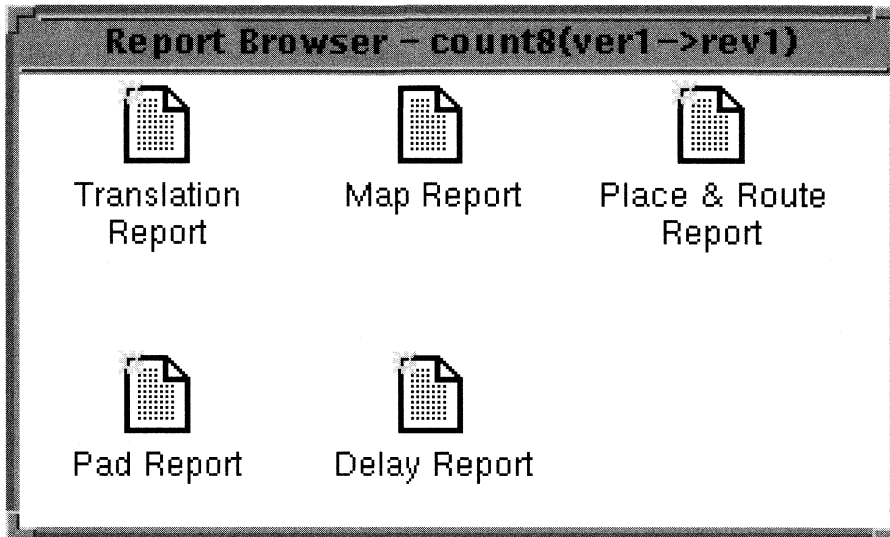


**Figure 3-9 Place and Route Stage Completed**

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 to make sure that no errors were encountered.

8. Select **Utilities** → **Report Browser** to invoke the Report Browser.



**Figure 3-10 Reports Available After Place & Route Stage**

The three newly created reports are the Delay Report, Pad Report, and Place and Route Report.

- **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.
- **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 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.

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 a period constraint, it reports the worst path and notes any timing errors if any paths fail to meet the desired timing. 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 to 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 or Open to accept the physical constraints file.

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	Logical
Resource	Resource(s)			

```

-----
CLB_R32C32.XQ  Tcko  1.830R  n172  n171
CLB_R32C32.G4  net   6.563R  n171
CLB_R32C32.COUT Topcy  3.900R  n172dd_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 Tbycp  0.350R  n170dd_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 Tbycp  0.350R  n168dd_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  n166dd_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 8ns. Simply doubling the block delays would give 16ns, 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 8ns, 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 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.

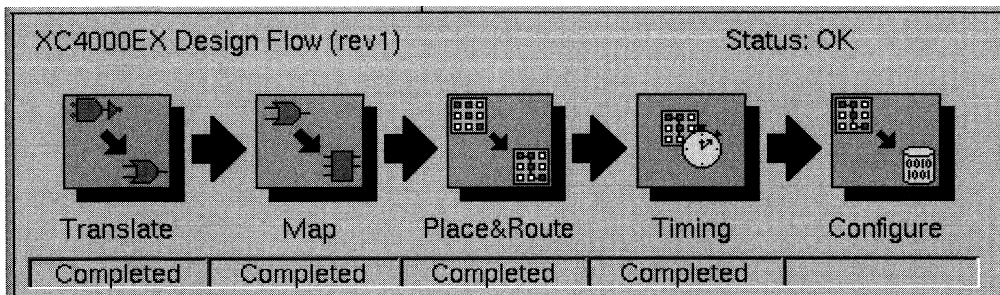
5. Click on the Edit Template button to the right of the Implementation template pulldown list box.

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 the “Completed Timing Stage” figure.

**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 3-11 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 following chapters in the *Development System Reference Guide*.

- “NGD2EDIF” chapter
- “NGD2XNF” chapter
- “NGD2VER” chapter

- “NGD2VHDL” chapter

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.

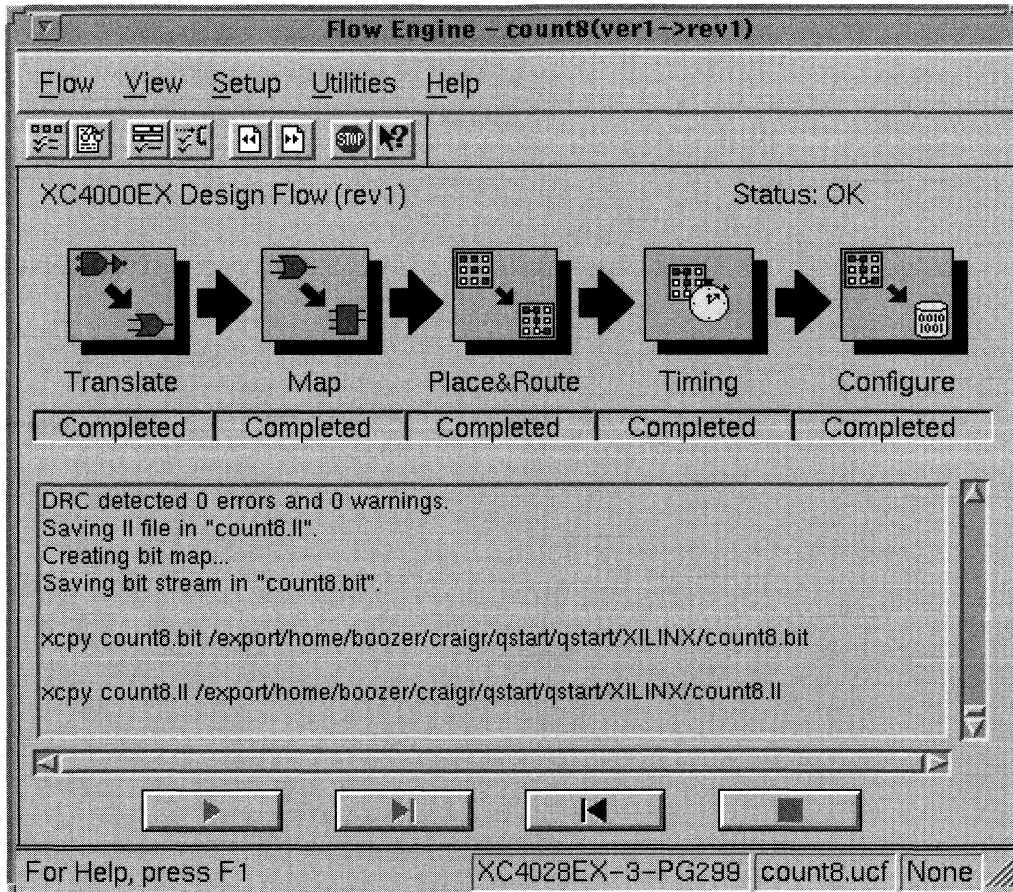
2. **Select Setup** → **Options** to open the Options dialog.
3. Click on the Edit Template button to the right of the Configuration template pulldown list box. The XC4000 Configuration Options dialog is displayed.
4. Click on the CMOS radio boxes in the Threshold Levels area for both inputs and outputs.
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.il`. The first file (`count8.bit`) is the actual configuration data while the second file (`count8.il`) is the logical allocation file used to determine where the probable points in the design are.



The *design.11* file is used for performing device readback using the Hardware Debugger. For more information on device readback, see the "Saving and Loading Readback Data" section of the *Hardware Debugger User Guide*.



**Figure 3-12 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.

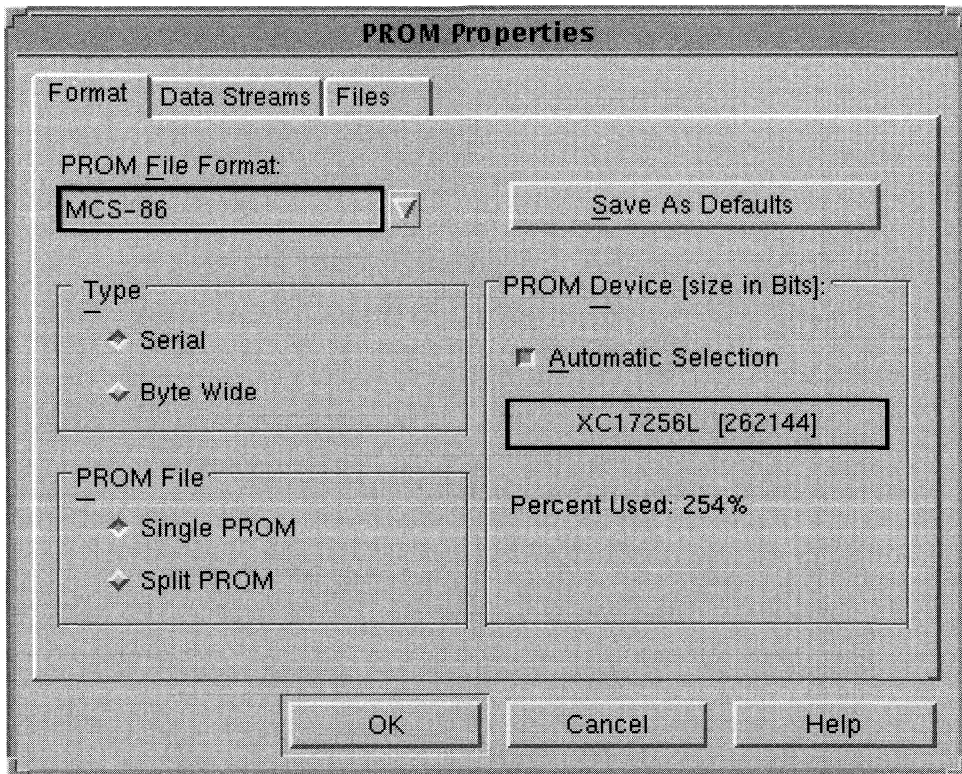
1. To invoke the PROM File Formatter, click on the PROM File Formatter icon in the Toolbox.

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.

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 “Configuring Multiple Device Daisy Chains from One Parallel PROM” section of 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 configuration 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.

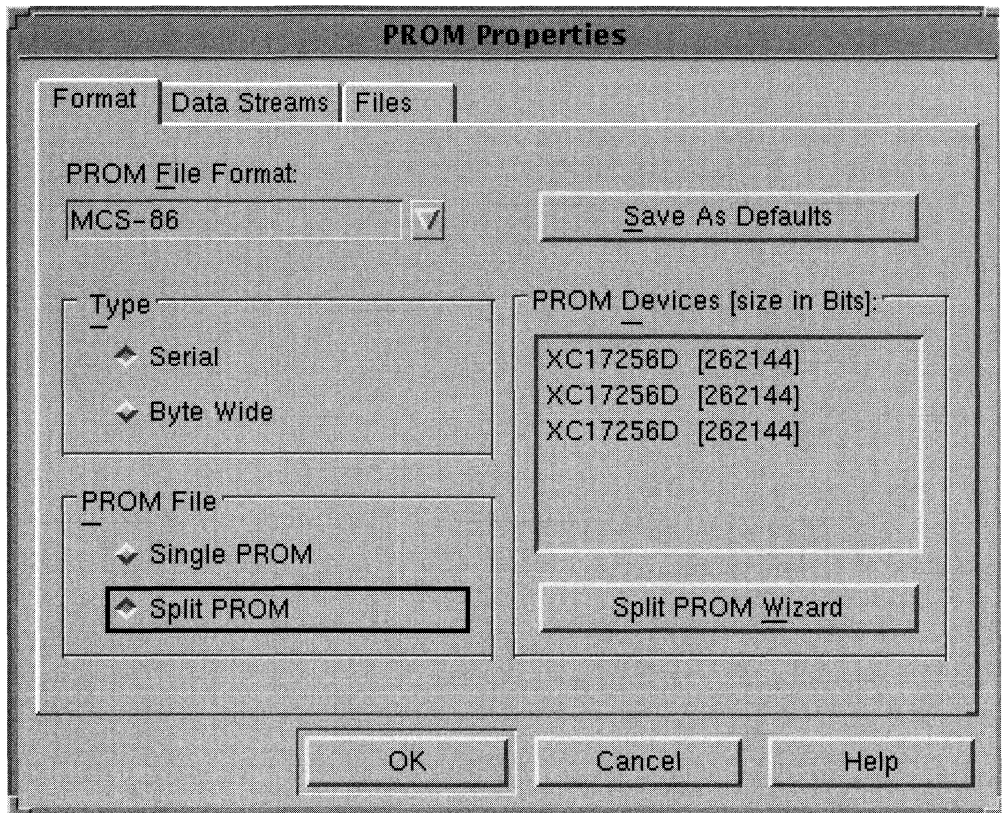
2. Select **File** → **PROM Properties** to open the PROM Properties dialog.



**Figure 3-13 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 3-14 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.



## 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.

- “Starting Xilinx Tools” section
- “Selecting Options” section
- “Using Constraint Files” section
- “Guiding An Implementation” section
- “Static Timing Analysis” section
- “Creating Simulation Files” section
- “Downloading A Design” section
- “Multi-Pass Place and Route” section

Refer during discussion to “M1 Software Design Flow” figure and “Detailed Design Flow” figure. The “M1 Software Design Flow” figure provides an overview of the M1 Software Design Flow; and the “Detailed Design Flow” figure provides 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
```

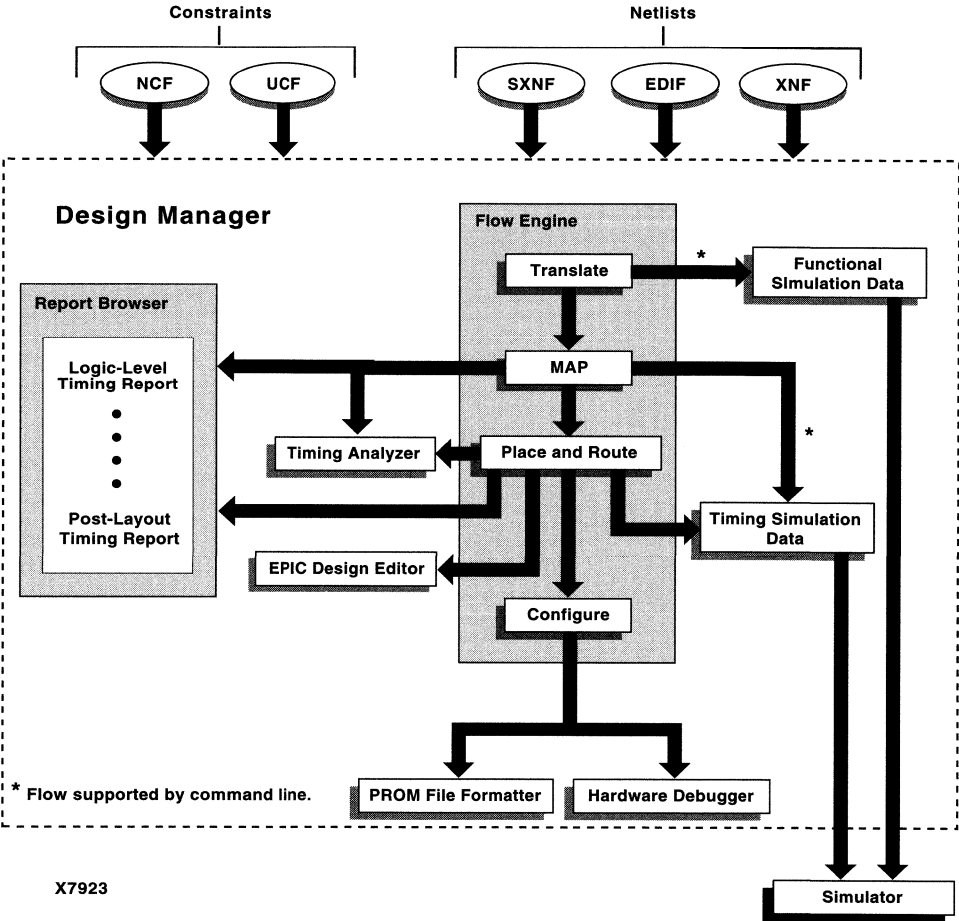
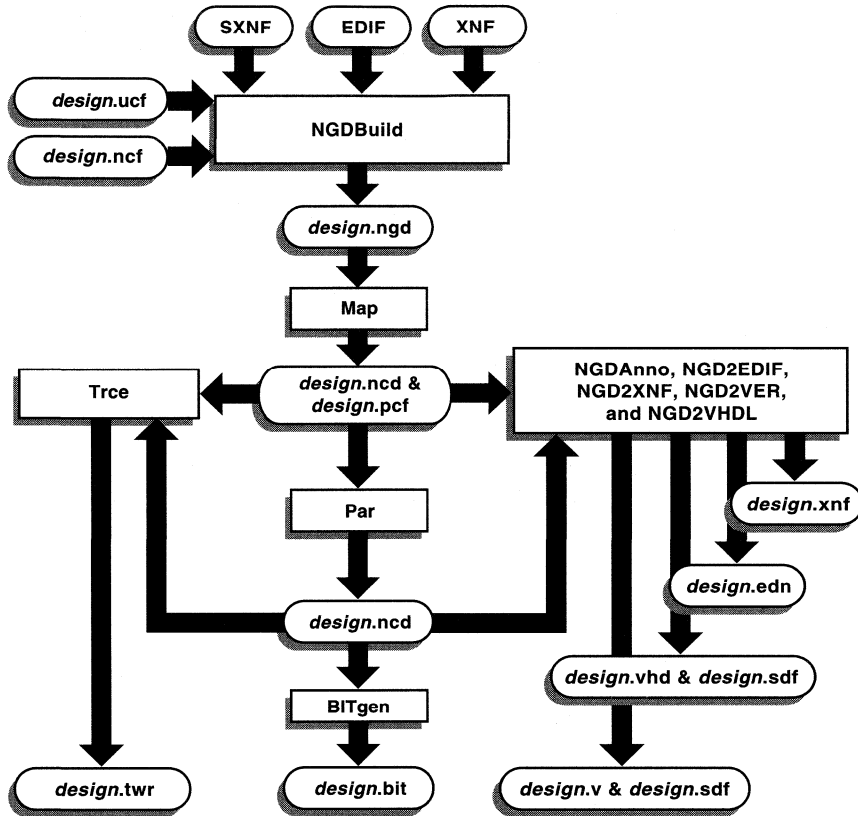


Figure 4-1 M1 Software Design Flow





X8037

Figure 4-2 Detailed Design Flow

## Creating A Project

From Design Manager (refer to “Design Manager Menu” figure), select **File** → **New Project**. 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 following appropriate appendix.

- “Cadence Concept and Verilog Interface Notes” appendix
- “FPGA Express Interface Notes” appendix
- “Mentor Graphics Interface Notes” appendix
- “Synopsys Interface Notes” appendix
- “Viewlogic Interface Notes” appendix
- “LogiBLOX” appendix
- “Instantiated Components” appendix
- “M1 Constraints Guide” appendix

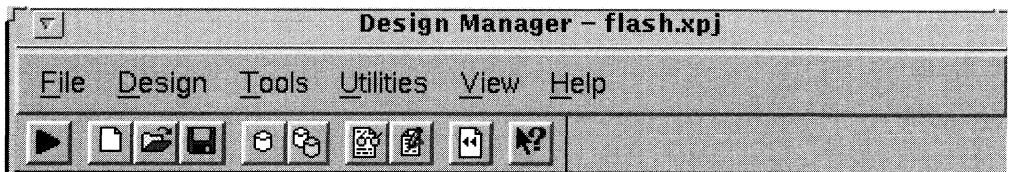


Figure 4-3 Design Manager Menu

## 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 can be 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

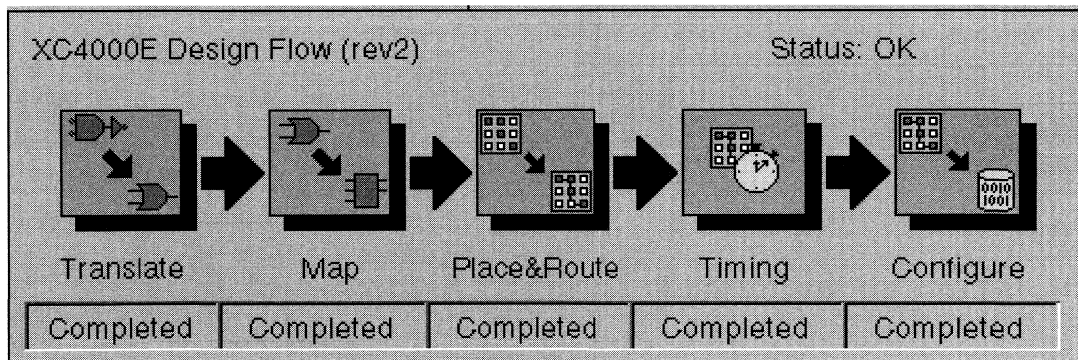
The Flow Engine’s first step, Translate, merges all of the input netlists. This is accomplished by running NGDBuild.

## Map

The next step is Map. Map optimizes the gates and trims unused logic in the merged NGD netlist. Map also maps the design's 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 Block (CLB) and input/output block (IOB) structures, are assigned to specific locations 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 "Flow Engine indicates completion of each design segment." figure.



**Figure 4-4** 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 the “Report Browser” figure.

### 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.

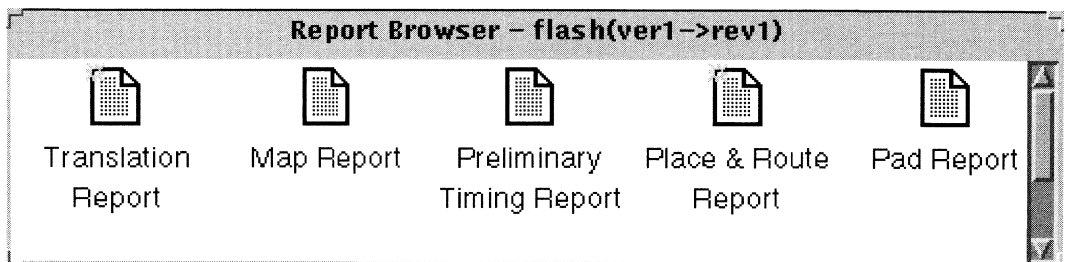


Figure 4-5 Report Browser

### 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 keyword 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.

- Design Score - The Design Score measures the relative goodness of the design. 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 Flows" section 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

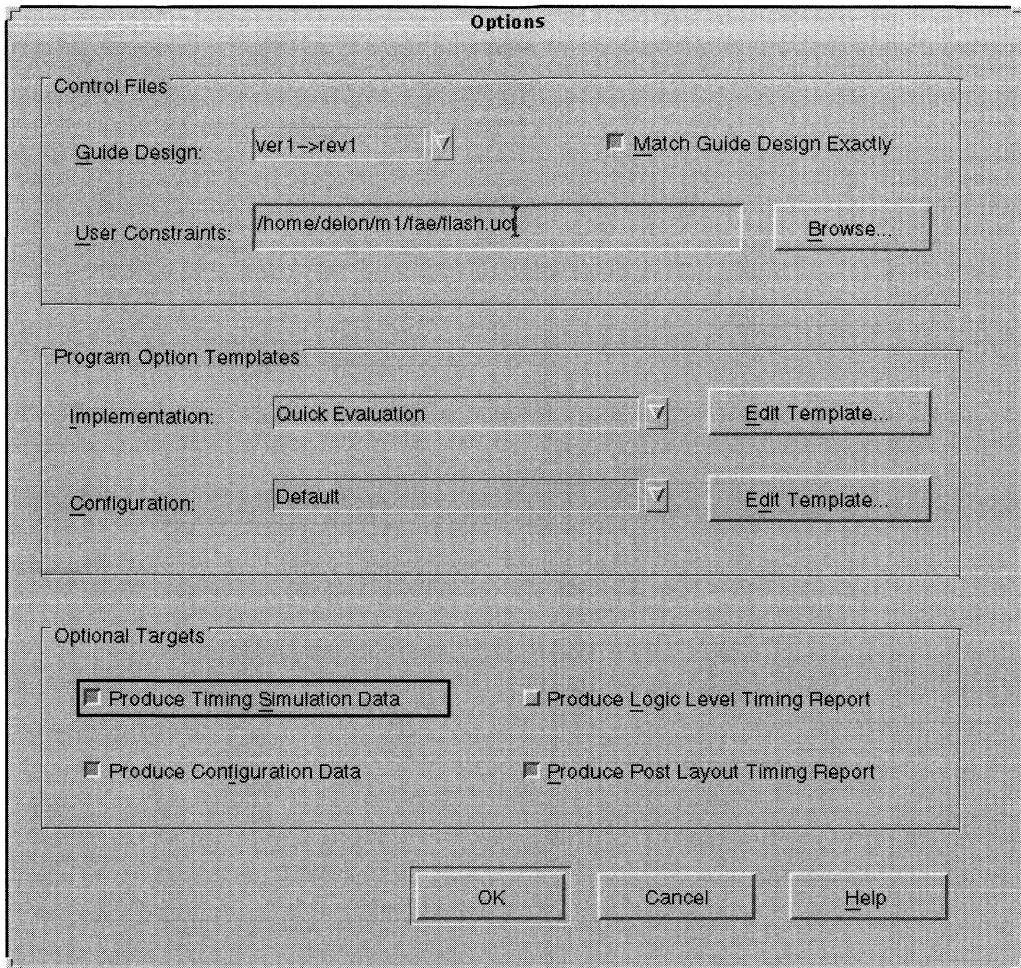
The Pad Report lists the design's pinout in three ways.

- 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 4-6 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`.

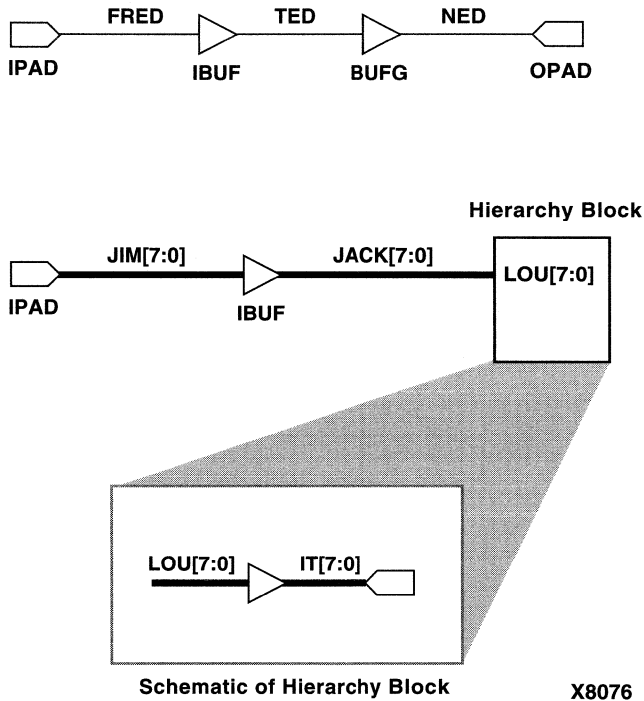
## Creating A User Constraint File

The User Constraint File (.UCF) is a user-created ASCII file that holds timing and location constraints. It is read by NGDDBuild 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>.

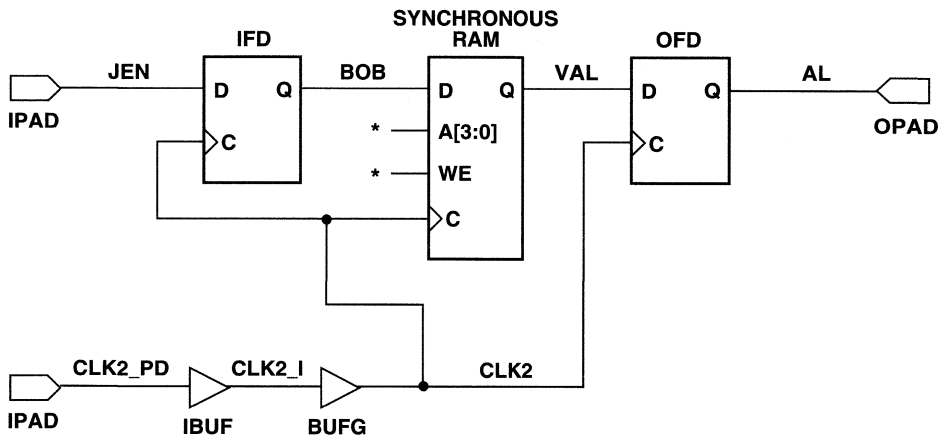
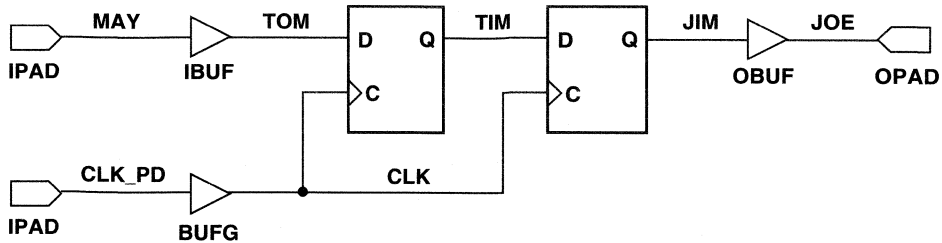
# Lock the input pins
```

```
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 logic delays on timing constraints, clock frequencies, and path delays. A post-place-and-route timing report, that incorporates both logic and routing delays, can be generated as a final evaluation of the design's timing constraints, clock frequencies, and path delays. Detailed timing constraint, clock, and path analysis for post-map or post-place-and-route implementations can be accomplished by using the interactive Timing Analyzer tool.

## **Static Timing Analysis After Map**

Post-map timing reports can be very useful in evaluating timing performance. 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.
  - Edit the template implementation
  - Select the timing tab
  - 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 NGDDBuild 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, most net delays are zero.

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

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

## Creating Timing Simulation Data

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 see the “NGDAnno” chapter of 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 NGD2XNF. 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 6.0 XNF 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 “Downloading Basics” section of 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 “Debugging Basics” section of 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.
- You can select the number of re-entrant routing passes to be done. If left in “Auto”, PAR will continue to perform routing iterations until either it determines that it is no longer making significant progress or the design constraints have been fully met.

You can also select the number of clean up passes to run. Two types of clean-up routing passes can be invoked, the Cost Based and the Delay Based. Clean up passes are run after the “main” routing passes are complete. The effectiveness of each type is dependent on the design, device, and constraints of the implementation. No predictable criteria can be suggested to choose one style over the other. The best methodology is to select no more than three passes for each (in most cases, a single pass for each is sufficient), and use the PAR report to determine which was most effective and try using more clean-up 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

---

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. This appendix contains the following sections.

- “Documentation” section
- “Setting up the Xilinx/Cadence Interface” section
- “Cadence/Verilog and M1 Design Flow” section
- “Setting Up for Concept” section
- “Using HDL Direct” section
- “Iterated Instances Vs. Size Support” section
- “Starting Up Concept” section
- “Functional Simulation” section
- “Translating a Design to Xilinx EDIF” section
- “Timing Simulation” section
- “Support for Board Level Simulation” section
- “Pin Locking” section
- “Timing Constraints” section

### 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:
$OPENWINHOME/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 “Cadence/Verilog Interface and M1 Design Flow” figure) 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.



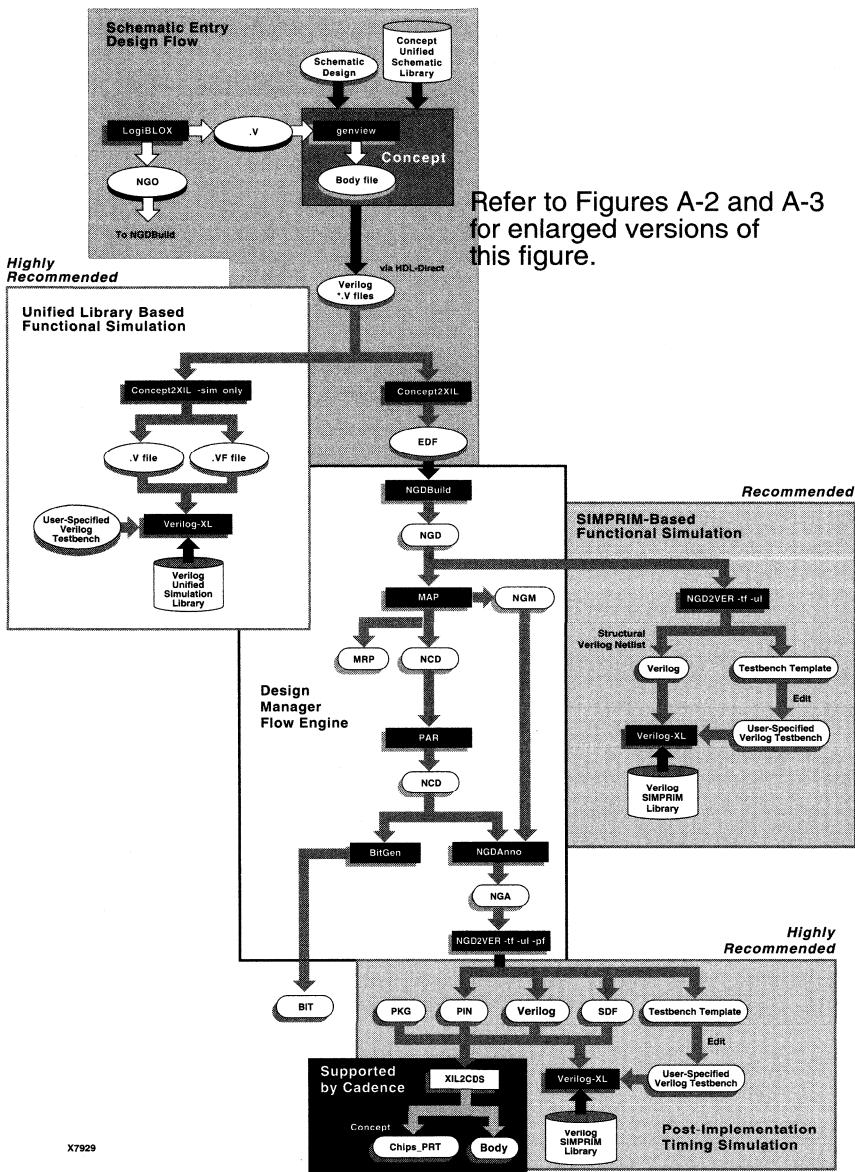


Figure A-1 Cadence/Verilog Interface and M1 Design Flow

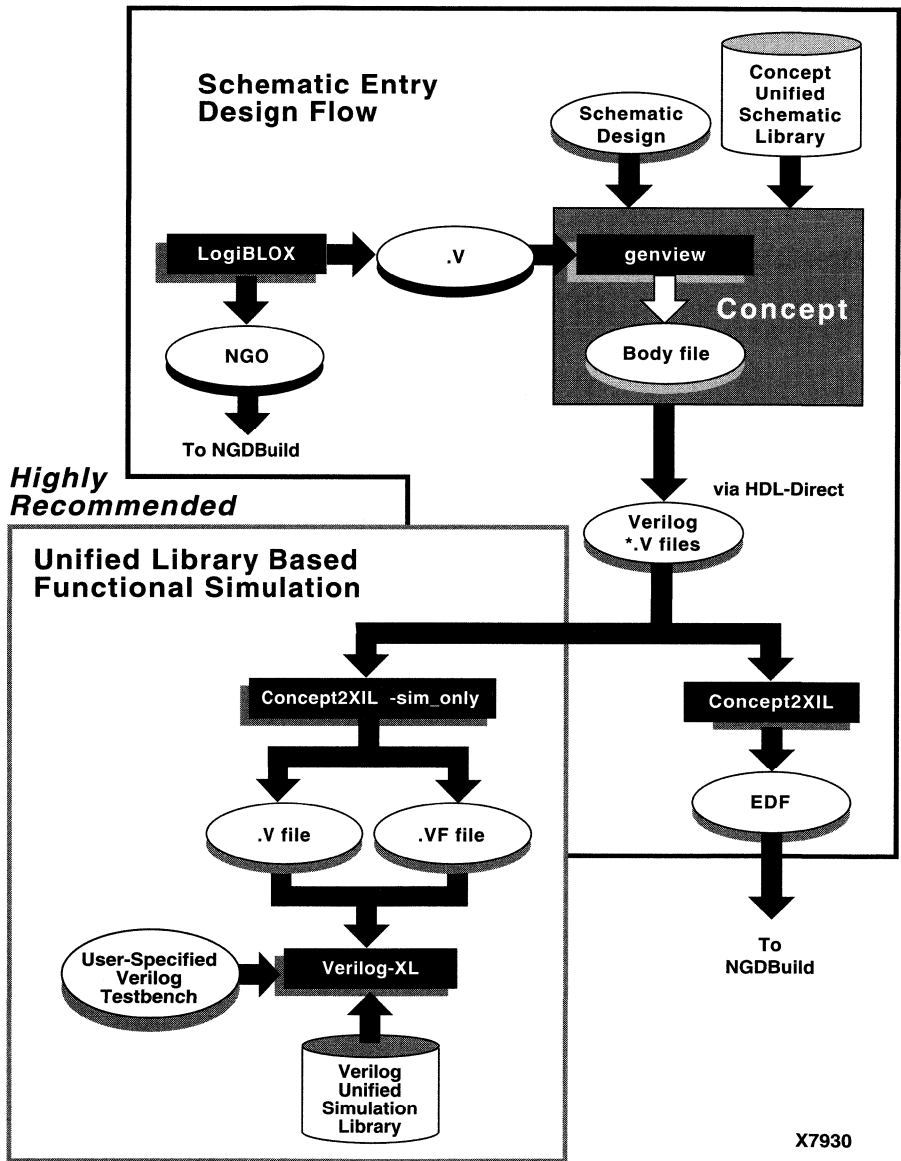
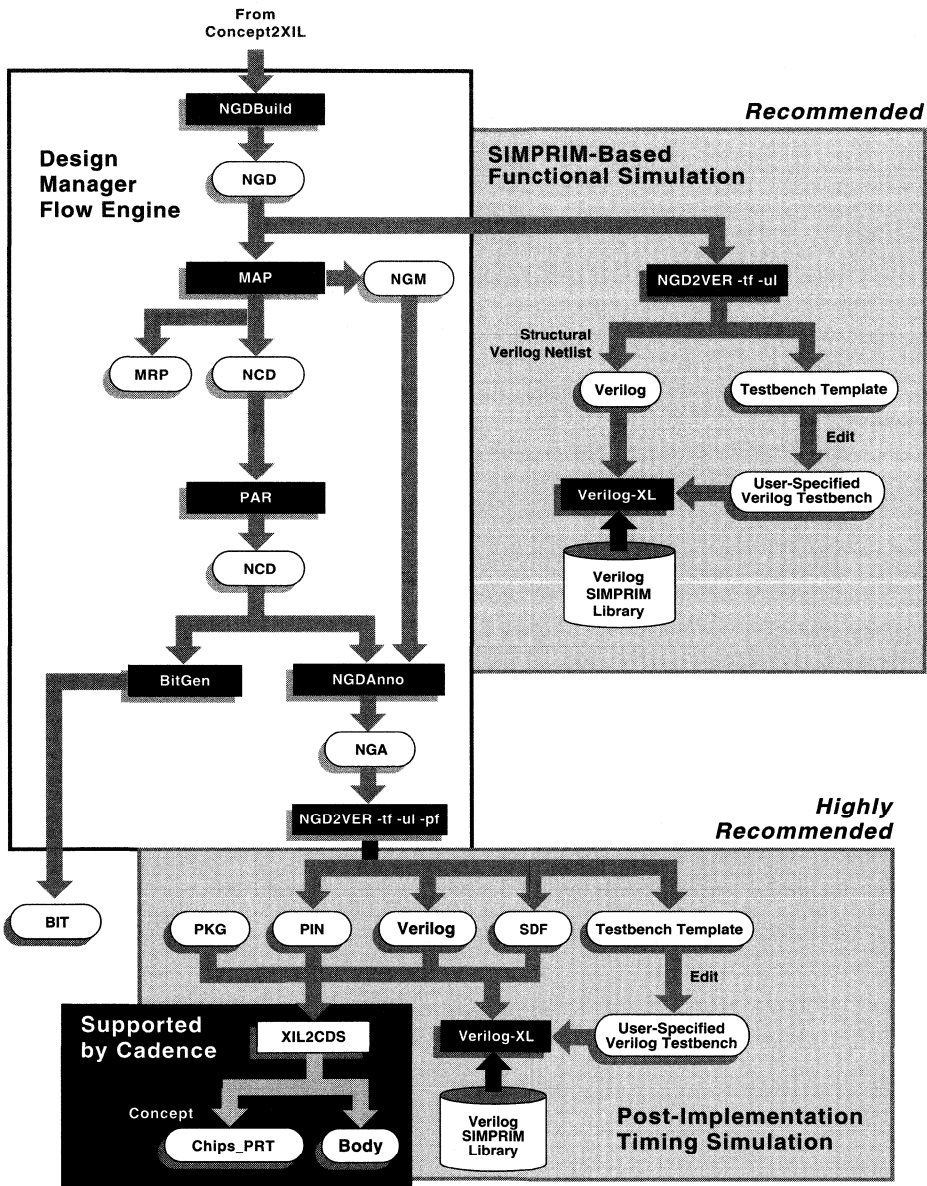


Figure A-2 Top Portion of Figure A-1



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 the design contains non-schematic blocks, then functional simulation may be performed after NGDDBuild is completed. All designs can be simulated at this point since non-schematic blocks, if any exist, are merged together by NGDDBuild.
- 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
set capslock_off

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 &
```

## Functional Simulation

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

In a netlist for a 4000E(X) design that uses STARTUP, the Global Set and Reset ("GSR") net that leads to every flip-flop is connected to the STARTUP block implicitly. Toggling the signal that controls the GSR pin is needed to begin simulation and to simulate resetting the device. Even if your design does not utilize the STARTUP block, the GSR line should be pulsed once at the beginning of simulation to .simulate the initial behavior of the device

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

```
`define GSR_SIGNAL testfixture.design.signal_on_GSR_pin
```

Here *testfixture* is the name of the testfixture module, *design* is the instance name of the instantiated design, and *signal\_on\_GSR\_pin* is the net name that sources the STARTUP GSR pin.

The signal that actually hooks up to the STARTUP GSR pin should be used; if the signal comes in from the IPAD with no other logic in between the IPAD and STARTUP primitive (such as an inverter), you may use the input signal name, if you wish. In your testfixture, you may proceed to assign values to the input pin that you use as your global reset (you do not have to assign values to the GSR pin signal itself). For example, assuming "global\_reset" is the name of the port controlling the GSR pin on the STARTUP symbol.

```
module my_testfixture(input_net, output_net, global_reset);
`define GSR_SIGNAL my_testfixture.uut.signal_on_GSR_pin
.....
my_design uut (.in(input_net), .out(output_net), .rst(global_reset));
.....
initial begin
    global_reset=1;
    #300 global_reset=0;
    // assign inputs
```

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

```
`define GSR_SIGNAL test.GSR
reg test.GSR;
```

```
.....
.....
```

```
initial begin
```

```
    test.GSR=1;
    #300 test.GSR=0;
    // assign inputs now
```

- For the 5200 family (which has a STARTUP symbol available), use

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

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

- For the 9500 family, use.

```
`define PRLD_SIGNAL test.PRLD
reg PRLD
//no 9K STARTUP, so use this dummy
```

```
.....
.....
```

```
initial begin
```

```
    test.PRLD=1;
    #300 test.PRLD=0;
    // assign inputs now
```

- For the 3000A family, use.



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

and use a similar procedure as described above for the 9500. Note GR on a 3000A is active-low.

## 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_testfixture.stim  
my_design.v -f my_design.vf
```

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

## Post-NGDBuild Functional Simulation

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

```
concept2xil -family target_tech design_name  
ngdbuild -p part_name design_name  
ngd2ver -ul -tf design_name.ngd  
verilog +delay_mode_unit test_fixture.stim design_name.v
```

## Translating a Design to 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<sup>\*\*\*</sup>) where <sup>\*\*\*</sup> = target architecture.

## 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 *-ul* option causes NGDSVER to automatically ass a “uselib” directive to the *.V* file that references the Xilinx-supplied Verilog SIMPRIM libraries.

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

If you re-use your functional simulation test fixture file and your design contains a STARTUP block and the GSR (and/ or GTS) pin on the STARTUP block is connected to a signal, you will need to make one modification to the test fixture--the GSR\_SIGNAL (and/ or GTS\_SIGNAL) macro(s) must be commented out.

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

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

To run the simulation, type.

```
verilog my_testfixture.stim my_design.v
```

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

## Support for Board Level Simulation

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

Typical syntax.

```
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*.

## FPGA Express Interface Notes

---

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 vM1.3. Only the core tools and a third party simulation tool are needed in addition to FPGA Express to fully create and simulate a design. This chapter contains the following sections.

- “Installation of FPGA Express” section
- “Design Entry With FPGA Express” section
- “Simulation With FPGA Express” section
- “Documentation” section
- “FPGA Express/XC4000EX Pre-Release Design Flow” section
- “Timing Constraints” section
- “Porting Code from FPGA Compiler to FPGA Express” section

## 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 (95) version 7.0 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 NGDDBuild or the Design Manager.

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

1. behavioral
2. post-NGDDBuild
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.

## 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 “FPGA Express/M1 Design Flow” figure. 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*.

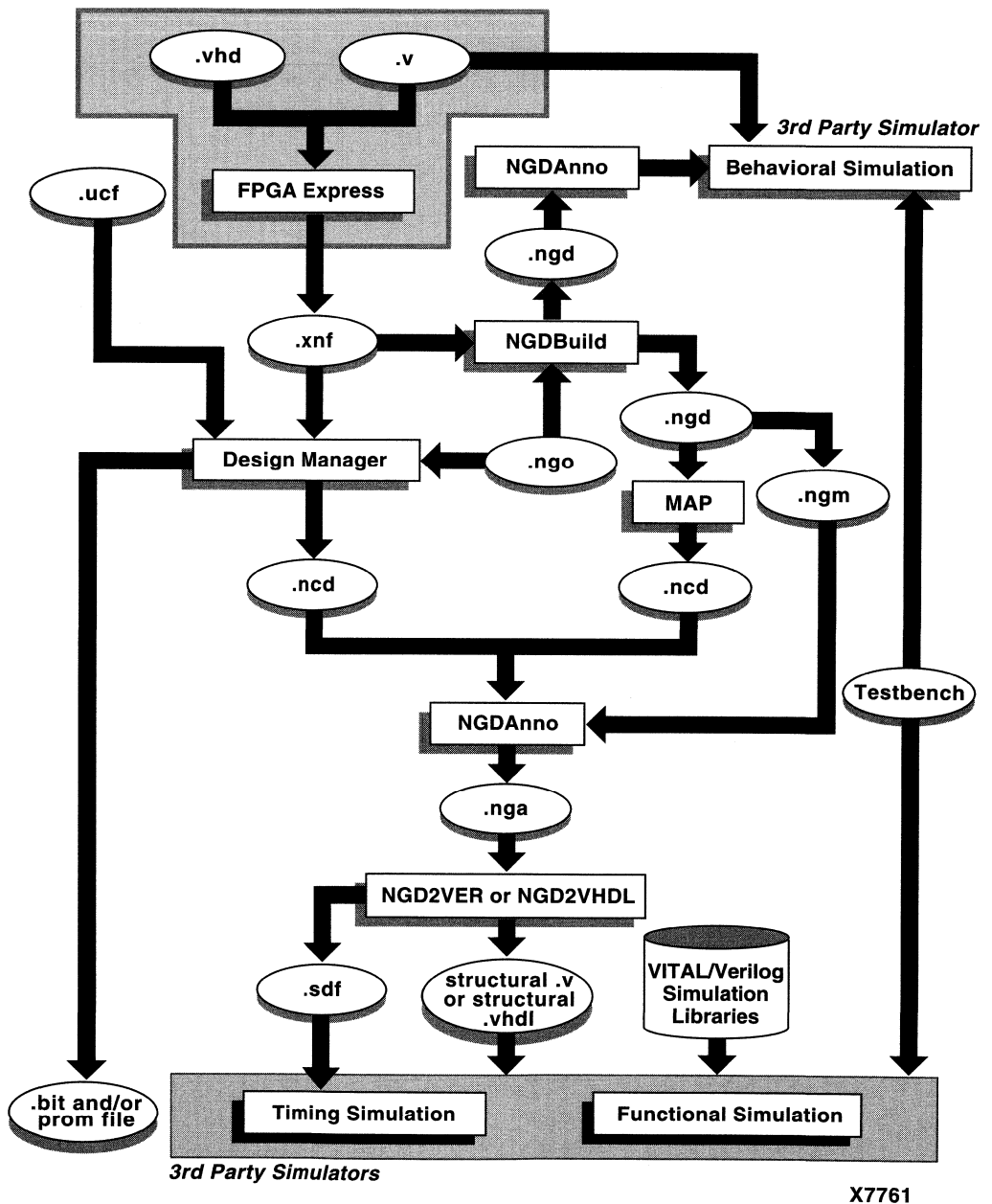


Figure B-1 FPGA Express/M1 Design Flow

# Appendix C

## Mentor Graphics Interface Notes

---

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. This chapter contains the following sections.

- “Documentation” section
- “Setting up the Xilinx/Mentor Interface” section
- “Mentor/M1 Software Design Flow” section
- “Translating a Design to Xilinx EDIF” section
- “Timing Simulation” section
- “Mentor-Related Environment Variables” section
- “Library Locations and Sample MGC Location Map” section
- “Pin Locking” section
- “Timing Constraints” section

## 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)
- SIMPRIMS (add)
- MGC\_GENLIB (add)
- MGC\_LOCATION\_MAP (add)
- path (modify)
- LD\_LIBRARY\_PATH (modify for SunOS/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 SIMPRIMS $LCA/simprims
```

---

```
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 Chapter 1 of this manual.

## Mentor/M1 Software Design Flow

Refer to “Mentor/M1 Software Flow” figure 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).
- 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.

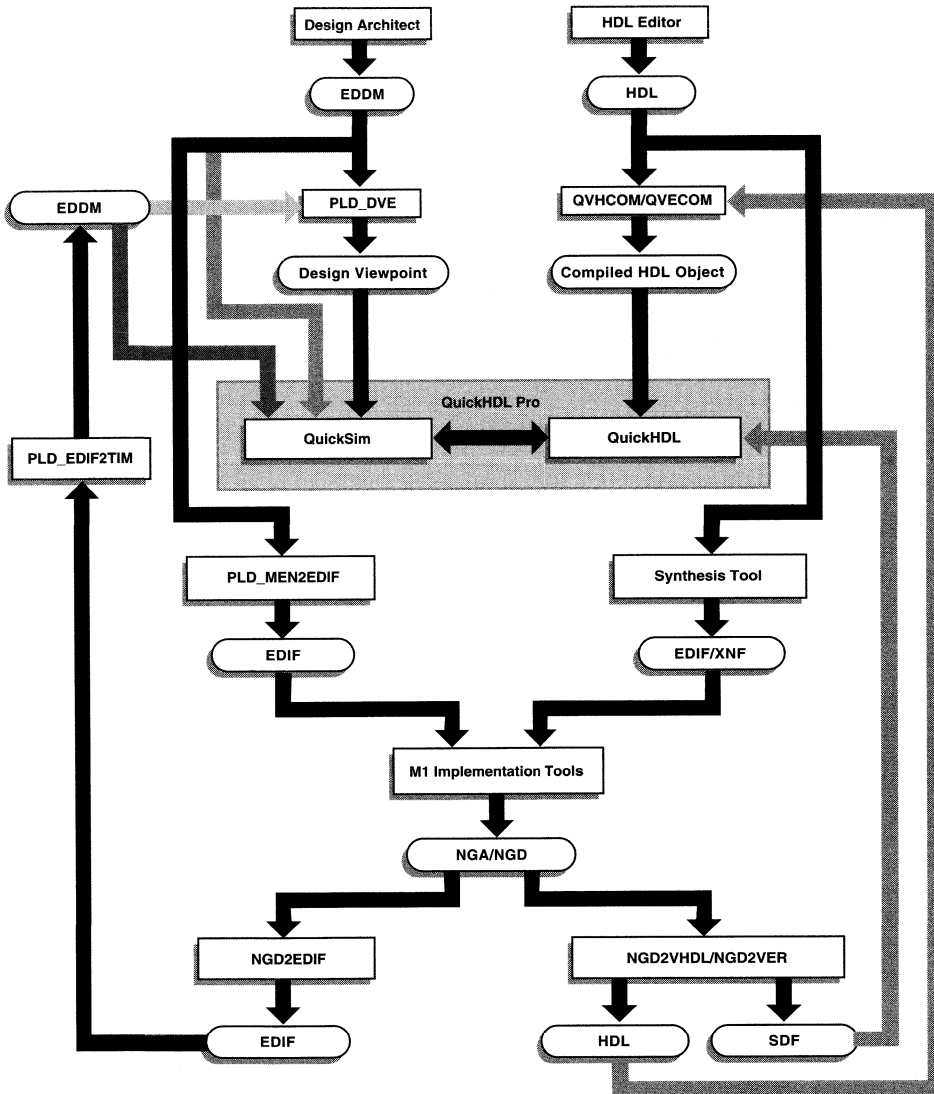
```
pld_dve -s my_design xc4000ex
```

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

To simulate this design, run.

```
pld_quicksim my_design
```

This runs QuickSim for functional simulation without cross-probing



X8094

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” section or the “Manual Translation” chapter of the *Mentor Graphics Interface/Tutorial 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” section or the “Manual Translation” chapter of the *Mentor Graphics Interface/Tutorial Guide*.

## Timing Simulation

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

### Generating a Timing-Annotated EDIF Netlist

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

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

This creates an EDN file compatible with the Mentor interface.

### Generating a Timing Model

After creating the EDN file, run PLD\_EDIF2TIM to generate a timing model with the following command.

```
pld_edif2tim my_design.edn
```

This creates an EDDM-type component under my\_design\_lib/my\_design.



---

## 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.

```
pld_dve -s 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 remove the `-cp` option to run without cross-probing.)

```
pld_quicksim my_design_lib/my_design -cp -  
tim_typ_  
-consm_messages
```

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

You may also use the `PLD_EDIF2TIM` and `PLD_QuickSim` icons in `PLD_DMGR`. For more information on timing simulation, including a more detailed explanation on cross-probing, see the “Timing Simulation” section or the “Manual Translation” chapter of the *Mentor Graphics Interface/Tutorial 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 Interface/Tutorial Guide* 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 Interface/Tutorial Guide*.

## Pin Locking

Pad symbols (IPAD, OPAD, etc.) have generic pin-location (“LOC”) properties already attached to them. (They appear as “PXX” on the pad symbol.) You can place pads in specific locations on the device by modifying these properties as required. (An example property value for a pad symbol may be “P24”.) Note that “bused” pad symbols (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

---

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. This chapter contains the following sections.

- “Documentation” section
- “Setting Up the Xilinx/Synopsys Interface” section
- “Synopsys/M1 Software Design Flow” section
- “Examples of Synopsys Setup Files” section
- “Timing Constraints and DC2NCF” section
- “FPGA Compiler Users” section
- “Entity Coding Examples” section

## 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 Chapter 1, 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*.

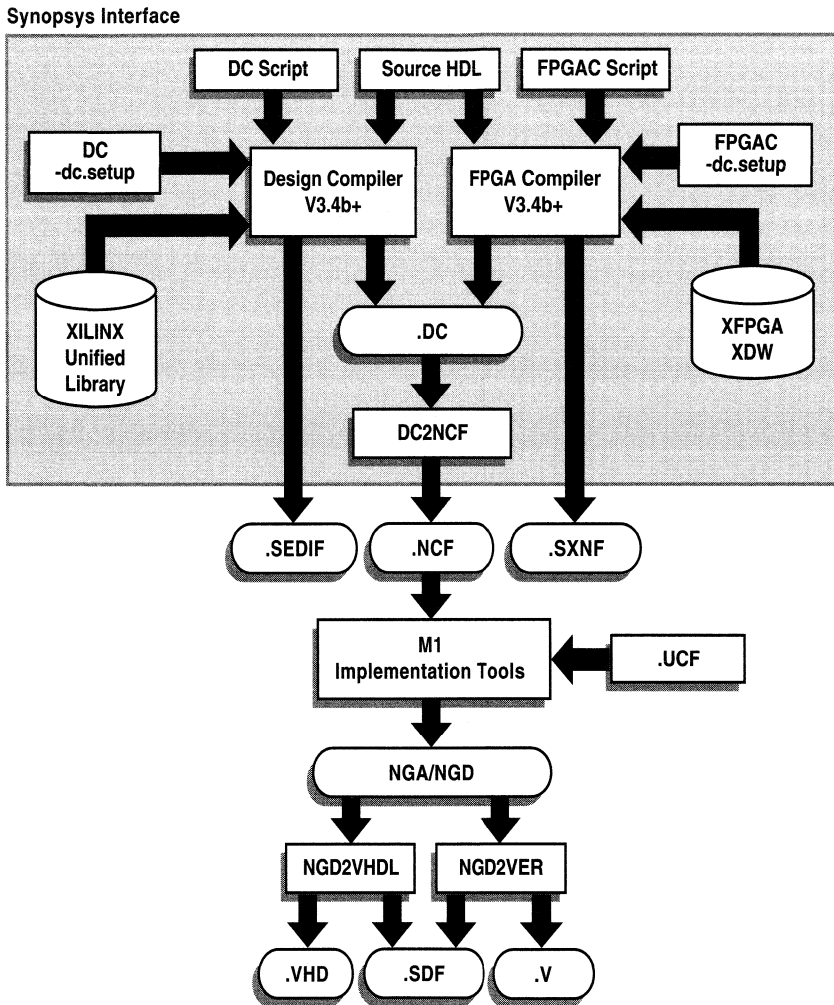
## Examples of Synopsys Setup Files

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 vhd1 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 -slewrates 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

---

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. This chapter contains the following sections.

- “Documentation” section
- “Setting Up Viewlogic Interface on Workstations” section
- “Setting Up Xilinx/Viewlogic Interface on the PC” section
- “Viewlogic/M1 Software Design Flow” section
- “Setting Up Project Libraries” section
- “Assigning a Pin Location” section
- “Using Special XC4000EX Features” section

### 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 “Viewlogic/M1 Core Technology Design Flow” figure 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.

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

- A ViewSim netlist file (.VSM) must be created for simulation. This file is created from WIR files that have come directly from the Viewlogic design entry tools, or from the Xilinx 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.

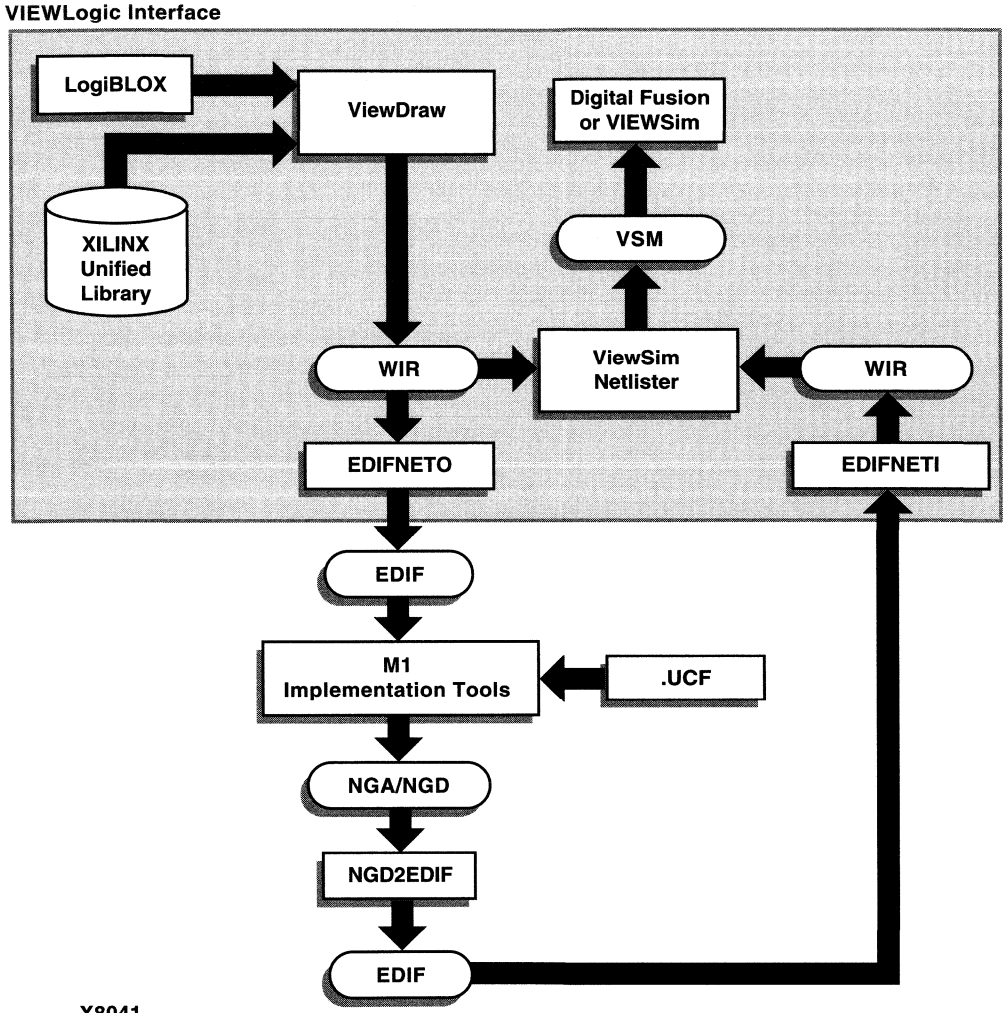


Figure E-1 Viewlogic/M1 Core Technology Design Flow

## 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/Tutorial 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 “LogiBLOX” appendix 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

---

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, XC4000L, XC4000EX, XC4000XL and XC5200.

This chapter contains the following sections.

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

## 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 three ways.

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

**lbgui**

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

## Using LogiBLOX for Schematic Design

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

### 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 (*.vhi*) or verilog (*.vei*) design file to the top level design. Then, specify the design names in the component instantiation section.

### Analyzing a LogiBLOX Module

Before starting behavioral simulation on an instantiated LogiBLOX module, the LogiBLOX library has to be analyzed. The following



three sections list the commands that can be used in Mentor, Synopsys, and Viewlogic to analyze the library.

## Mentor QuickHDL

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

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

## Instantiated Components

---

This appendix lists the components most frequently instantiated in synthesis designs. The function of each component is briefly described and the pin names are supplied, along with a listing of the Xilinx product families involved. 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*. This chapter contains the following sections.

- “STARTUP Component” section
- “BSCAN Component” section
- “READBACK Component” section
- “RAM and ROM” section
- “Global Buffers” section
- “Fast Output Primitives” section
- “IOB Components” section

### 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 7-1 Startup Library Component**

Name	Family	Description	Outputs	Inputs
STARTUP	4000E/L, 4000EX, 4000XL, 5200/L <sup>1</sup>	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, 5200/L <sup>1</sup>	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, 5200/L	Connects to the BSCAN TDI input. Loads instructions and data on each low-to-high TCK transition.	I	—

**Table G-2 BSCAN Library Components**

Name	Family	Description	Outputs	Inputs
TDO	4000E/L, 4000EX, 4000XL, 5200/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, 5200/L	Connects to the BSCAN TMS input. It determines which boundary scan is performed.	I	—
TCK	4000E/L, 4000EX, 4000XL, 5200/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 the *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

**Table G-3 Readback Library Components**

Name	Family	Description	Outputs	Inputs
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

**Table G-4 RAM and ROM Library Components**

Name	Family	Description	Outputs	Inputs
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
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**

<b>Name</b>	<b>Family</b>	<b>Description</b>	<b>Outputs</b>	<b>Inputs</b>
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
BUFGP <sup>1</sup>	4000E/L, 4000EX, 4000XL	A primary global buffer, distributes high fan-out clock, or control signals throughout PLD devices.	O	I
BUFGS <sup>2</sup>	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 × 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**

<b>Name</b>	<b>Family</b>	<b>Description</b>	<b>Outputs</b>	<b>Inputs</b>
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



## M1 Constraints Guide

---

This appendix discusses some of the more common constraints you can apply to your design to control the timing and layout of a Xilinx FPGA or CPLD. For a complete listing of all supported constraints, please refer to the “Attributes, Constraints, and Carry Logic” chapter of the *Libraries Guide*. For a more complete discussion of how timing constraints work in M1, refer to the “XACT-Performance Utility” chapter of the *Development System Reference Guide*. This appendix contains the following sections.

- “Constraint Entry Mechanisms” section
- “Translating and Merging Logical Designs” section
- “Constraints File Overview” section
- “UCF Timing Constraints” section
- “Layout Constraints” section
- “Efficient Use of Timespecs and Layout Constraints” section
- “Standard Block Delay Symbols” section
- “Table of M1-Supported Constraints” section
- “Basic UCF Syntax Examples” section
- “Constraining LogiBLOX RAM/ROM with Synopsys” section

## Constraint Entry Mechanisms

The M1 version of the Xilinx Alliance and Foundation design tools allow users to control the implementation of a design through constraints which affect the mapping and layout of the physical circuit. Additionally, a user must specify the “path” timing requirements of the circuit to obtain the best results, and allow the imple-

mentation tools to determine the layout which satisfies these requirements.

The various design constraints available for use within M1 can be entered at design creation (i.e., the logical domain) or after the design is mapped (i.e., the physical domain). Constraints entered in the logical domain are either entered into the schematic, or applied to a synthesis process and then forward annotated through the netlisting mechanism (DC2NCF for Synopsys FPGA and Design Compiler flows, XNF file generation for the Synopsys FPGA Express flow).

Constraints entered in the physical domain are entered directly into the Physical Constraints File (PCF). These constraints are conceptually the same as those entered during design creation, however they are directly related to objects within the physical design database, and are therefore applied using the PCF syntax.

“Constraint Entry Flow” figure illustrates the constraints entry mechanisms of the M1 version of the Xilinx tools. The rest of this appendix describes the steps involved in implementing a design within the M1 tools, with specific focus on how constraints are entered at each stage of design processing.

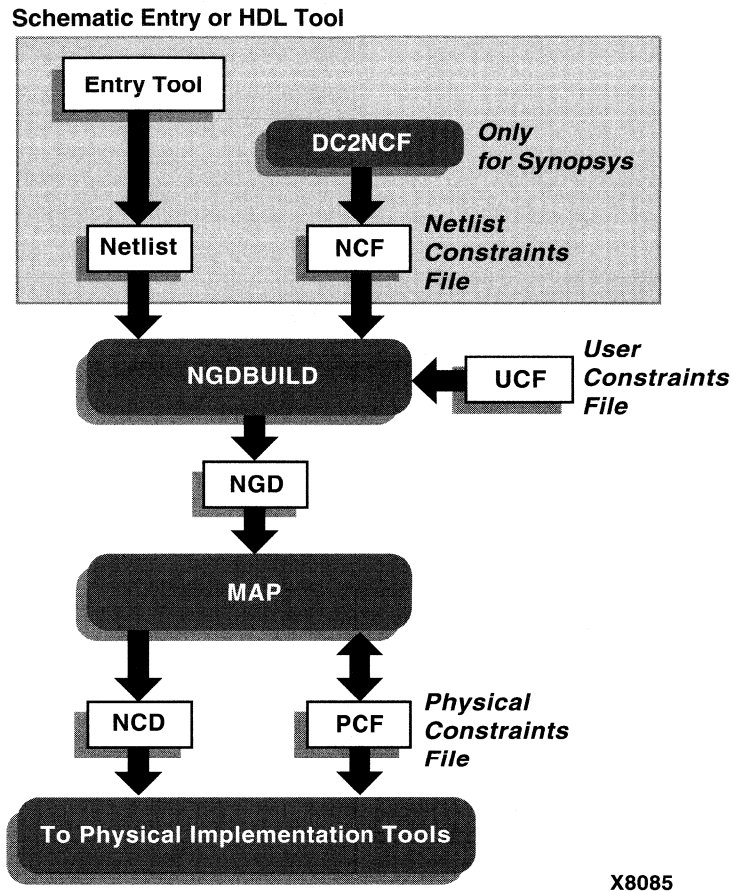


Figure H-1 Constraint Entry Flow

## Translating and Merging Logical Designs

The process of implementing a design within the M1 tools starts by constructing a logical design file (NGD) that represents the design created by the NGDBuild application (as shown in “Constraint Entry Flow” figure).

The NGD file contains all of the design's logic structures (gates) and constraints. The NGD file is produced through the NGDBuild process which controls the translation and merging of all of the related logic design files. All design files are translated from industry standard netlists into intermediate NGO files by one of two netlist translation programs XNF2NGD or EDIF2NGD. The exception to this rule is logic which is created through the use of LogiBlox components. With some tools, such as Viewlogic, LogiBLOX components may be compiled directly in memory from within the tool, and are therefore never written to disk as a separate intermediate NGO file. If LogiBLOX is invoked separately, it will write out an NGO file.

## Constraints File Overview

### The Netlist Constraint File (NCF)

The Netlist Constraint File was developed as an alternative means for third party vendors to provide the M1 tools with design constraints. Historically, these constraints are in the design netlist and are annotated to the equivalent elements within the design's NGO file. In M1, translating a logic design netlist to a NGO file includes annotating constraints present in the NCF file to the NGO design elements. The NCF file is local in scope and therefore must have the same root name as the netlist being translated. In addition, the local scope of the netlist allow entries within the NCF file to refer to specific nets or symbols without prefixing the entire hierarchical path of the net or symbol.

“Constraint Entry Flow” figure illustrates the Synopsys workstation design flow which utilizes a NCF file. The program DC2NCF converts the Synopsys constraints into NCF syntax. Refer to the “XSI Design Flow” section of the *Synopsys (XSI) Interface /Tutorial Guide* for detailed information on using this flow.

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



## The User Constraint File (UCF)

The User Constraint File was developed to provide users an easy mechanism for constraining a logical design without returning to the design capture tools. The process of building the complete logical design representation (NGD files) is the job of NGDBuild. In developing this complete design database, NGDBuild annotates design constraints which are present within a UCF file. The syntax for the UCF constraints file is identical to the syntax of the Netlist Constraints File (NCF).

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

Another difference is that UCF constraint file may override the NCF constraints. Because the Netlist Constraint File (NCF) is considered an alternative mechanisms for design creation packages to pass constraints to the M1 implementation tools, no conflicts should occur. In contrast, the User Constraint File (UCF) annotation includes a resolution mechanism which allows for UCF constraints to overwrite constraints present in the NCF. UCF constraints are considered more significant due to their later appearance in the design flow, and provide a mechanism for establishing or modifying logical design constraints without requiring the user to re-enter a schematic or synthesis tool.

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

## The Physical Constraints File (PCF)

The NGD, or physical representation of the design, describes the design in terms of FPGA resources. The layout and timing analysis tools work with the NCD representation. The PCF contains the

constraints as they relate to the NCD. Layout and timing constraints are written in terms of the physical design's components (COMPs), fractions of COMPs (BELs) and collections of COMPs (macros). Because of this different design viewpoint, the PCF syntax is not necessarily the same as the logical design constraint files (UCF/NCF). Furthermore, because the PCF file is written for use by the physical implementation tools, logical names may no longer be similarly represented in the physical design. Some of the advantages of working with constraints in the PCF file are.

- It is readily modified and immediately applicable to NCD (i.e., there is no need to re-run NGDBuild or MAP in order to run layout or analysis tools).
- Implications of the mapping of logical design structures into the physical structures of the FPGA only become obvious once the design is evaluated using the physical tools. Altering the PCF file for "what-if" analysis can be desirable.

If you modify the PCF file, you should be certain that you enter your constraints after the "SCHEMATIC END ;" line. Otherwise, your constraints will be overwritten every time MAP is re-executed.

## Case Sensitivity

Users should be aware that the M1 constraints are case sensitive, since EDIF is a case-sensitive format. Therefore, you should always be sure that you specify the net names and instance names exactly as they are in your schematic or code. You should also be consistent when using TNMs and other user-defined names in your constraints file; always use the same case throughout. For site names (such as "CLB\_R2C8" or "P2"), you should use upper case only.

## UCF Timing Constraints

### The "From:To" Style Timespec

When using the From:To style of constraint, the path(s) that are constrained are specified by declaring the start point and end point, which must be of type pad, flip-flop, latch, RAM, or user-specified sync point (see TPSYNC). To group a set of endpoints together, you may attach a TNM attribute to the object (or to a net that is an input to the object). A TIMEGRP is a mechanism for combining two or more

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

TNMs are used by the M1 tools in the same way as the XACT 5.2 tools—identification of a group of design objects which are to be referenced within a Timespec. If a TNM is placed on a net, the M1 tools determine TNM membership by tracing forward from the specified net to all the valid endpoints of the net. Refer to the “XACT-Performance Utility” chapter of the *Development System Reference Guide* for more information on this subject.

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

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

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

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

The first line of the above example illustrates the application of the TNM (Timing Name) *PIPEA* to the net named *DATA\_EN*. The second line illustrates the TIMEGRP design object formed using a pattern matching mechanism in conjunction with the pre-defined TIMEGRP “PADS”. In this example, the TIMEGRP named *BUSPADS* will include only those PADs with names that start with *BUS*.

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

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

There is an additional keyword that can be added to the From:To spec that allows the user to narrow the set of paths that are covered. By

using the From:Thru:To form, you may constrain only those paths that go through a certain set of nets, defined by the TPTHU keyword, as shown in the following example.

```
# UCF FROM:TO Timespec using THRU

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

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

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

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

## Using TPSYNC

In the XACT 5.x/6.x methodology, only flip-flops, RAMs, latches and pads could be identified as a startpoint or endpoint for a timing specification. However, in the M1 toolset, you may now define any node as a source or destination for a Timespec with the TPSYNC keyword. The use of TPSYNC is similar to TPTHU—it is a label that is attached to a set of nets, pins, or instances in the design.

For instance, suppose a design has a PAD ENABLE\_BUS that must arrive at the enable pin of several different 3-state buffers in less than a specified time. With the M1 tools, you can now define that 3-state buffer as an endpoint for a timing spec. For example:

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

NET BUS3STATE TPSYNC=bus3;
TIMESPEC TSNewSp3=FROM:PAD(ENABLE_BUS):TO:bus3:20ns;
```

In the NET statement shown above, the TPSYNC is attached to the output net of a 3-state buffer called *BUS3STATE*. If a TPSYNC is attached to a net, then the source of the net is considered to be the endpoint (in this case, the 3-state buffer itself). The subsequent TIMESPEC statement can use the TPSYNC name just as it uses a TNM name.

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

```
# Note TPSYNC attached to 3-state buffer's enable PIN
PIN $1I6/BUSMACRO1/TRIBUF34.T TPSYNC=bus1;
TIMESPEC TSNewSpc1=FROM:PAD(ENABLE_BUS):TO:bus1:20ns;
```

In this example, the instance name of the 3-state buffer is given followed by the pin name of the enable (.T). If a TPSYNC is attached to a primitive input pin, then the primitive's input is considered the startpoint or endpoint for a timing specification. If it is attached to a output pin, then the output of the primitive is used.

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

```
# Note TPSYNC attached to 3-state buffer INSTANCE
INST $1I6/BUSMACRO2/BUFFER_2 TPSYNC=bus2;
TIMESPEC TSNewSpc2=FROM:PAD(ENABLE_BUS):TO:bus2:20ns;
```

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

## The Period Style Timespec

In addition to From:To style Timespecs, there are several other forms of Timespecs provided by the M1 tools. Of particular significance is the PERIOD Timespec. The example below illustrates the use of the PERIOD Timespec referenced to a timegroups *CLK2\_GRP* and *CLK3*.

```
# UCF PERIOD style Timespecs
NET CLK2 TNM = CLK2_GRP ;
NET CLK3 TNM = CLK3 ;
TIMESPEC TS03 = PERIOD CLK2_GRP 50 ;
TIMESPEC TS04 = PERIOD CLK3 TS03 * 2 ;
```

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

The PERIOD constraint covers all timing paths which start or end at a register, latch, or synchronous RAM which is clocked by the referenced net. The only exception to this rule are paths to output pads which are not covered by the PERIOD constraint. (Input pads, which

are the source of a “pad-to-setup” timing path for one of the specified synchronous elements, are covered by the PERIOD constraint.)

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

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

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

NET CLK PERIOD = 40 ;
```

## The Offset Constraint

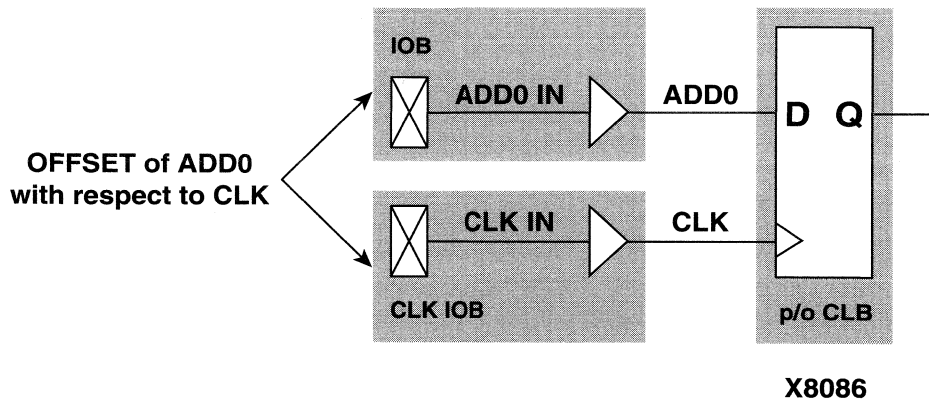
The OFFSET constraint is applied to a net connecting with a PAD (see “Using OFFSET Constraints” figure). It defines the delay of a signal relative to a clock, and is only valid for registered data paths. The OFFSET constraint specifies the signal delay external to the chip, allowing the implementation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

```
# Net form of the OFFSET timing constraint

NET ADD0_IN OFFSET = IN 14 AFTER CLK ;
```

In analyzing OFFSET paths, the M1 timing tools adjust the PERIOD associated with the constrained synchronous element based on both the timing specified in the OFFSET constraint and the delay of the referenced clock signal. In “Using OFFSET Constraints” figure, assume a delay of 8 ns for the signal CLK to arrive at the CLB, a 5 ns setup time for ADD0, and a 14 ns OFFSET delay for the signal ADD0. Assume a period of 40 ns is specified. The M1 tools allocate 29 ns for

the signal ADD0 to arrive at the CLB input pin ( $40\text{ns} - 14\text{ns} + 8\text{ns} - 5\text{ns} = 29\text{ns}$ ).



**Figure H-2 Using OFFSET Constraints**

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

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

## Ignoring Paths

When a Timespec is declared that includes paths where the timing is not important, the tools may create a less optimal route since there is more competition for routing resources. This problem can be alleviated by using a TIG (timing ignore) attribute on the non-critical nets.

TIG will cause all paths that fan out from the net or pin where it is applied to be ignored for purposes of timing analysis.

```
#TIG example
```

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

The above syntax indicates that \$1I456/slow\_net should not have the Timespec TS01 or TS04 applied to it.

## Controlling Skew

A new feature of the M1 tools is the ability to control the maximum allowed skew. The maximum skew (MAXSKEW) is the difference between the longest and shortest driver to load connection delays for a given net.

```
#MAXSKEW example
```

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

The above indicates that 3 ns is the maximum skew allowed on *net\_a*.

## Constraint Precedence

A user may assign a precedence to Timespecs only within a certain class of constraints. For example, a user may specify a priority for a particular From:To specification to be greater than another, but may not specify a From:To constraint to have priority over a TIG constraint. The following example illustrates the explicit assignment of priorities between two same-class timing constraints, the lowest number having the highest priority.

```
# Priority UCF Example
```

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

“Precedence of Constraints” table illustrates the order of precedence for constraint files and timing constraints.

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



<b>Across Constraint Sources</b>	
<i>Highest Priority</i>	Physical Constraint File (PCF)
	User Constraint File (UCF)
<i>Lowest Priority</i>	Input Netlist / Netlist Constraint File (NCF)
<b>Within Constraint Sources</b>	
<i>Highest Priority</i>	TIG (Timing Ignore)
	FROM:USER1:THRU:USER_T: TO:USER2 Specification (USER1 and USER2 are user-defined groups)
	FROM:USER1:THRU:USER_T: TO:FFS Specification or FROM:FFS:THRU:USER_T:TO: :USER2 Specification (FFS is any pre-defined group)
	FROM:FFS:THRU:USER_T:TO: :FFS Specification
	FROM:USER1:TO:USER2 Specification
	FROM:USER1:TO:FFS Specifi- cation or FROM:FFS:TO:USER2 Specifi- cation
	FROM:FFS:TO:FFS specifica- tion
	Period specification
<i>Lowest Priority</i>	"Allpaths" type constraints

**Table H-1 Precedence of Constraints**

## Layout Constraints

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

```
# Mapping Constraint
    INST GLUE OPTIMIZE = SPEED ;

# Layout Constraint
# LOC a pin
    NET IOBLOCK/DATA0_IN LOC = P12 ;

# Reserve a pin (Pin 24 should be unused)
    CONFIG PROHIBIT = P24 ;
```

The layout constraint in the example above illustrates the use of a full hierarchical path name for the net named DATA0\_IN in the application of the I/O location constraint. In this example, IOBLOCK is a hierarchical boundary which contains the net DATA0\_IN. Location constraints applied to “Pad nets” are used to constrain the location of the PAD itself, in this case to site P12.

## Converting a Logical Design to a Physical Design

The process of mapping translates a design from the logical design domain to the physical design domain. The MAP process creates both the physical design components (CLBs, IOBs, etc. ) and the physical design constraints (layout and timing). The physical design components are written into a Native Circuit Description (NCD) file. The physical design constraints are written into a Physical Constraints File (PCF).

As the design flow of “Constraint Entry Flow” figure shows, MAP not only writes a PCF file, but also reads a specified pre-existing PCF file. MAP reads an existing PCF file in order to facilitate the over-riding of constraints that are contained within another logic design

using the “last one wins” resolution mechanism provided by the PCF file. The following paragraphs briefly describe this approach.

## Last One Wins Resolution

MAP creates new physical design constraints each time it converts a logical design into a physical design. The constraints which are created during this process are written into the “Schematic” section of the PCF file. This section is recreated each time MAP is run based on the constraints that are contained within the NGD file. The schematic section is always written at the top of the PCF file, and constraints that are in the PCF file but outside of the Schematic section (after the line “SCHEMATIC END”) are considered to be in the “User” section of the PCF file. The user section is read, syntactically checked, and rewritten each time MAP is run. Since these constraints always follow those written into the schematic section, they will always take precedence (following the “last-one-wins” rule).

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

## XC5200 Constraints

There are some special considerations for constraints for the XC5200 family.

- The XC5200 family requires that some constraints (such as LOC and RLOC) specify logic cell name within the CLB. For instance, the following constraint will LOC the instance *ISYM52* to the lowest logic cell of the CLB in row 5 and column 2.

```
INST ISYM52 LOC = CLB_R5C2.LC0 ;
```

If this was an XC4000 design, an extension would be optional (the XC4000 family does not use the LCx notation; it uses FFX and FFY to specify which flop and F and G to specify which function generator).

- The XC5200 family does not have flip-flops in the IOB, so two new constraints have been provided: INREG and OUTREG. PAR will attempt to place a register with a INREG attribute near the IOB that drives its Din pin, so it can use fast routes. OUTREG will cause PAR to attempt to place a register near the IOB that Qout sources.

```
INST near_input_flop INREG ;  
INST near_output_flop OUTREG ;
```

## Efficient Use of Timespecs and Layout Constraints

The previous section described the mechanisms available for constraining a design's timing within the M1 tools. The sections that follow summarize each of the constraints that are available for use. The natural question which will arise is "How should I describe my requirements to the tools?"

The robust nature of the language enables a designer to define their design requirements at the highest level of abstraction first, and then fine tune the timing requirements using more specific TIMESPECS as necessary. This is the methodology that should be followed.

The following observations help to illustrate the reasons why this methodology should be followed (from a tool runtime perspective).

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

In conclusion, overall design runtime is improved when a "qualified global" timing methodology is employed instead of a "thorough-detailed" timing methodology.

## The "Starter Set" of Timing Constraints

The following examples clearly identify the "preferred" mechanism for controlling the timing of your design. The preferred method assumes a goal of getting the required results in the fastest run-time possible. If the design has a single clock and required I/O timing which equals the clock period, all that is needed are the three constraints shown in the following example.

```
# Global UCF Example  
NET CLK1 PERIOD = 40 ;  
NET OUT* OFFSET = OUT 13 AFTER CLK ;
```

```
TIMESPEC TS01 = FROM PADS TO PADS 40 ;
```

If you need to account for extra delay external to the FPGA, then you might add the following:

```
NET INPUT* OFFSET = IN 8 BEFORE CLK ;
```

The PERIOD constraint covers all pad-to-setup and clock-to-setup timing paths. The OFFSET constraint covers the clock-to-pad timing for each of the output nets beginning with *OUT*. The OFFSET constraint accounts for the delay of the Clock Buffer/Net in the I/O timing calculations.

The PCF snippet below illustrates the differences in syntax between the NCF/UCF and PCF languages. In addition to the syntactical changes, it is important to note that net and instance names may change. As an example, one of the net matches resulting from the UCF "NET *OUT\**" constraint is now applied to "COMP *OUT1\_PAD*". The name *OUT1\_PAD* is the name assigned to the pad instance. In addition to name changes, another difference to note is the verbosity of the PCF. In the PCF there is additional syntax for "MAXDELAY", "TIMEGRP" and "Priority". These are all optional qualifications of the Timespec within the UCF, but written explicitly to the PCF file illustrating the full flexibility of the language.

```
# Global PCF example

SCHEMATIC START;

. . .

NET PERIOD "CLK_IN" = 40 nS HIGH 50.00% ;
COMP "OUT1_PAD" OFFSET = OUT 40 ns AFTER COMP "CLK";
COMP "OUT2_PAD" OFFSET = OUT 40 ns AFTER COMP "CLK";
COMP "INPUT1_PAD" OFFSET = IN 8 ns BEFORE COMP "CLK";

TS01 = MAXDELAY FROM TIMEGRP "PADS" TO TIMEGRP "PADS"
40000 pS PRIORITY 0;

SCHEMATIC END;
```

The next UCF example illustrates the use of both global constraints (PERIOD, OFFSET) for generally constraining the design, and detailed TimeSpecs (FROM:THRU:TO) for providing fast and slow exceptions to the general timing requirements. Because the amount of constraints placed on a design directly effect run time, it is recommended that users first apply global constraints, then apply individual constraints only to those elements of the design which require

additional constraints (or an exception to a constraint). The more global the constraints, the better the runtime performance of the tools.

```
# Sample UCF file
# Specify target device and package

CONFIG PART = XC4010e-PQ208-3;

# Global Constraints

NET CLK1 PERIOD = 40;
NET DATA_OUT* OFFSET = OUT 15 AFTER DCLK;
TIMESPEC TS01 = FROM PADS TO PADS 40;

# Layout Constraints

NET SCLINF LOC = P125;

# Detailed Constraints
# Exception to cover X_DAT and Y_DAT buses
NET ?_DAT* OFFSET = OUT 25 AFTER CLK_IN;

# Ignore timing on reset net

NET RESET_N TIG;

# Slow exception for data leaving INA FFs
TIMESPEC TS02 = FROM FFS(INA*) TO FFS 80;

# Faster timing required for data leaving RAM
TIMESPEC TS03 = FROM RAMS TO FFS 20;

# Form special time groups related to RAMs

INST $1I64 TNM = SPDRAM;
NET RAMBUS0 TPTHRU = RAMVIA;
NET RAMBUS1 TPTHRU = RAMVIA;

# Specify timing for this special timing path
TIMESPEC TS04 = FROM SPDRAM THRU RAMVIA TO FFS 45;
```

## Standard Block Delay Symbols

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

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

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

```
SCHEMATIC END;

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

ENABLE = ram_d_o;
```

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

**Table H-2 Timing Symbols and Their Default Values**

## Table of M1-Supported Constraints

For further explanation and examples of each of the constraints, please see “Attributes, Constraints, and Carry Logic” chapter of the *Libraries Guide*.

Constraint	M1.0			
	Schematic	NCF	UCF	PCF
ADD	Y	N	N	N
BASE	Y	N	N	N
BLKNM	Y	Y	Y	N
BUFG	Y	Y	Y	N
COLLAPSE	Y	Y	Y	N
CONFIG	Y	N	N	N
Config. Constraints	Y	Y	Y	N
D_INVERT	Y	N	N	N
DECODE	Y	Y	Y	N
DIVIDE1_BY DIVIDE2_BY	Y	Y	Y	N
DOUBLE	Y	N	N	N
DROP_SPEC	N	Y	Y	Y**
EQUATE_F EQUATE_G	Y	N	N	N
FAST	Y	Y	Y	N
FILE	Y	N	N	N
HBLKNM	Y	Y	Y	N
HU_SET	Y	Y	Y	N
INIT	Y	Y	Y	N
IO	Y	Y	Y	N
KEEP	Y	Y	Y	N
LOC =	Y	Y	Y	Y**



\*\* Use cautiously — while constraint is available there are syntax differences.

Constraint	M1.0			
	Design	NCF	UCF	PCF
LOWPWR=(ON   OFF)	Y	N	N	N
MAP	N	N	N	N
MAXDELAY	Y	Y	Y	Y**
MAXSKEW	Y	Y	Y	Y**
MEDDELAY	Y	Y	Y	N
MINIM	Y	N	N	N
Net Flag Attributes				
	F	Y	N	N
	H	Y	N	N
	P	N	N	Y
	S	Y	N	N
	X	Y	N	N
NODELAY	Y	Y	Y	N
NOREDUCE	Y	Y	Y	N
OFFSET	N	Y	Y	Y**
OPT	Y	N	N	N
OPTIMIZE	Y	Y	Y	N

\*\* Use cautiously — while constraint is available there are syntax differences.

Constraint	M1.0			
	Design	NCF	UCF	PCF
OPT Effort	Y	Y	Y	N
PART	Y	Y	Y	N
PERIOD	Y	Y	Y	Y**
PROHIBIT	Y	Y	Y	Y**

PWR_MODE	Y	Y	Y	N
REG	Y	N	N	N
RLOC	Y	Y	Y	N
RLOC_ORIGIN	Y	Y	Y	N
RLOC_RANGE	Y	Y	Y	N
SLOW	Y	Y	Y	N
TIG	Y	Y	Y	Y**
Timegroup Attributes	Y	Y	Y	N
TNM	Y	Y	Y	Y
TPSYNC	Y	Y	Y	N
TPTHURU	Y	Y	Y	N
TSidentifier = IGNORE	Y	N	N	N
Other TSidentifier	Y	Y	Y	Y**
U_SET	Y	Y	Y	N
USE_RLOC	Y	Y	Y	N
WIREAND	Y	Y	Y	N

\*\* Use cautiously — while constraint is available there are syntax differences.

## Basic UCF Syntax Examples

### PERIOD TIMESPEC

The PERIOD spec covers all timing paths that start or end at a register, latch, or synchronous RAM which are clocked by the reference net (excluding pad destinations). Also covered is the setup requirement of the synchronous element relative to other elements (ex. flip flops, pads, etc...).

**Note:** The default unit for time is nanoseconds.

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

## FROM:TO TIMESPECS

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

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

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

## OFFSET TIMESPEC

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

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

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

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

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

## TIMING IGNORE

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

**Note:** The “\*” character is a wild-card which can be used for bus names. A “?” character can be used to wild-card one character.

- Ignore timing of net *reset\_n*:

```
NET : reset_n : TIG ;
```

- Ignore *data\_reg(7:0)* net in instance *mux\_mem*:

```
NET : mux_mem/data_reg* : TIG ;
```

- Ignore *data\_reg(7:0)* net in instance *mux\_mem* as related to a TIMESPEC named TS01 only:

```
NET : mux_mem/data_reg* : TIG = TS01 ;
```

- Ignore *data1\_sig* and *data2\_sig* nets:

```
NET : data?_sig : TIG ;
```

## PATH EXCEPTIONS

If your design has outputs that can be slower than others, you can create specific timespecs similar to this example for output nets named *out\_data(7:0)* and *irq\_n*.

```
TIMEGRP slow_outs = PADS(out_data* : irq_n) ;  
TIMEGRP fast_outs = PADS : EXCEPT : slow_outs ;  
TIMESPEC TS08 = FROM : FFS : TO : fast_outs : 22 ;  
TIMESPEC TS09 = FROM : FFS : TO : slow_outs : 75 ;
```

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

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

- TIMEGRP example.

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

- TNM attached to instance example.

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

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

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

```
NET ff_clock_enable_net TNM = slowffs ;
```

- Example of using “slowffs” timegroup, in a FROM:TO timespec, with either of the three timegroup methods shown above.

```
TIMESPEC TS10 = FROM : slowffs : TO : FFS : 100 ;
```

## MISCELLANEOUS EXAMPLES

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

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

- Prohibit IO pin C26 or CLBR5C3 from being used.

```
CONFIG PROHIBIT = C26 ;
CONFIG PROHIBIT = CLB_R5C3 ;
```

- Assign an OBUF to be FAST or SLOW.

```
INST obuf_instance_name FAST ;
INST obuf_instance_name SLOW ;
```

- Constrain the skew or delay associate with a net.

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

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

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

```
INST input_ff_instance_name MEDDELAY ;
INST input_ff_instance_name NODELAY ;
```

- Also, constraint priority in your .ucf file is as follows.

Highest

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

See the on-line documentation ([dtext](#) → [Library Reference Guide](#)) for additional timespec features or additional information.

## Constraining LogiBLOX RAM/ROM with Synopsys

In the M1 XSI HDL methodology, whenever large blocks of RAM/ROM are needed, LogiBLOX RAM/ROM modules are instantiated in the HDL code. With LogiBLOX RAM/ROM modules instantiated in

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

**How many primitives are inside a LogiBLOX RAM/ROM module?**

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

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

**How are the RAM primitives inside LogiBLOX RAM/ROM modules named?**

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

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

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

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

## Referencing a LogiBLOX Module/Component in the FPGA/Design Compiler and FPGA Express Flow

LogiBLOX RAM/ROM modules in the M1 FPGA/Design Compiler flow are constrained via a .ucf file. LogiBLOX RAM/ROM modules instantiated in the HDL code can be referenced by the full-hierarchical instance name. If a LogiBLOX RAM/ROM module is at the top-level of the HDL code, then the instance name of the LogiBLOX RAM/ROM module is just the instantiated instance name. In the case of a LogiBLOX RAM/ROM, which is instantiated within the hierarchy of the design, the instance name of the LogiBLOX RAM/ROM module is the concatenation of all instances which contain the LogiBLOX RAM/ROM. For FPGA/Design Compiler, the concatenated instance names are separated by a “/”. In the example, the RAM32X1S is named *memory*. The module *memory* is instantiated in Verilog module *inside* with an instance name U0. The module *inside* is instantiated in the top-level module test. Therefore, the RAM32X1S can be referenced in a .ucf file as U0/U0. For example, to attach a TNM to this block of RAM, the following line could be used in the .ucf file.

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

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

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

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

## Referencing the Primitives of a LogiBLOX Module in the FPGA/Design Compiler and FPGA Express Flow

Sometimes its necessary to apply constraints to the primitives that compose the LogiBLOX RAM/ROM module. For example, if you choose a floorplanning strategy to implement your design, it may be necessary to apply LOC constraints to one or more primitives inside a LogiBLOX RAM/ROM module. Consider the RAM32x2S example above, suppose that the each of the RAM primitives had to be

constrained to a particular clb location. Based on the rules for determining the MEMx\_y instance names, using the example from above, each of RAM primitives could be referenced by concatenating the full-hierarchical name to each of the MEMx\_y names. The RAM32x2S created by LogiBLOX would have primitives named MEM0\_0 and MEM1\_0. So, for FPGA/Design Compiler, CLB constraints in a .ucf file for each of these two items would be:

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

For FPGA Express, the CLB constraints would be:

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

## FPGA/Design Compiler and Express Verilog Example

### test.v:

```
module test (DATA, DATAOUT, ADDR, C, ENB);  
    input [1:0] DATA;  
    output [1:0] DATAOUT;  
    input [4:0] ADDR;  
    input C;  
    input ENB;  
    wire [1:0] dataoutreg;  
    reg [1:0] datareg;  
    reg [1:0] DATAOUT;  
    reg [4:0] addrreg;  
  
    inside U0  
        (.MDATA(datareg), .MDATAOUT(dataoutreg), .MADDR(addrreg)  
        ), .C(C), .WE(ENB));  
  
    always@(posedge C) datareg = DATA;  
    always@(posedge C) DATAOUT = dataoutreg;  
    always@(posedge C) addrreg = ADDR; endmodule
```

### inside.v:

```
module inside (MDATA, MDATAOUT, MADDR, C, WE);  
    input [1:0] MDATA;  
    output [10] MDATAOUT;
```



```

input [4:0] MADDR;
input C;
input WE;

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

endmodule

```

### **memory.v (FPGA/Design compiler only)**

```

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

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

endmodule

```

### **runscript (FPGA/Design compiler only)**

```

TOP=test part = "4028exp9299-3"
read -f verilog "guts.v"
read -f verilog "inside.v"
read -f verilog "test.v"
current_design TOP
remove_constraint -all
set_port_is_pad "*"
insert_pads
compile
write -format db -hierarchy -output TOP +
  "_compiled.db"
replace_fpga
set_attribute TOP "part" -type string part
write -format db -hierarchy -output TOP + ".db"
ungroup -all -flatten
write_script > TOP + ".dc" sh dc2ncf test.dc
remove_design guts
write -f xnf -h -o TOP + ".sxnf"

```

### **test.ucf (FPGA/Design compiler only)**

```

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

```

## test.ucf (FPGA Express only)

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

## FPGA/Design Compiler and Express VHDL Example

### test.vhd

```
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
use IEEE.STD_LOGIC_UNSIGNED.all;  
  
entity test is  
port( DATA: in STD_LOGIC_VECTOR(1 downto 0);  
      DATAOUT: out STD_LOGIC_VECTOR(1 downto 0);  
      ADDR: in STD_LOGIC_VECTOR(4 downto 0);  
      C, ENB: in STD_LOGIC);  
end test;  
  
architecture details of test is  
signal dataoutreg,datareg: STD_LOGIC_VECTOR(1 downto 0);  
signal addrreg: STD_LOGIC_VECTOR(4 downto 0);  
component inside  
  port(MDATA: in STD_LOGIC_VECTOR(1 downto 0);  
       MDATAOUT: out STD_LOGIC_VECTOR(1  
downto 0);  
       MADDR: in STD_LOGIC_VECTOR(4 downto 0);  
       C,WE: in STD_LOGIC);  
end component;  
begin  
  U0: inside port  
  map(MDATA=>datareg.,MDATAOUT=>dataoutreg.,MADDR=>addr  
reg,C=>C,WE=>ENB);  
  
  process( C )  
  begin  
    if(Cevent and C=1) then  
      datareg <= DATA;  
    end if;  
  end process;  
  
  process( C )  
  begin
```

```

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

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

end details;

```

### inside.vhd

```

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

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

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

end details;

```

### runscript (FPGA/Design compiler only)

```

TOP=test part = "4028exp299-3"
analyze -f vhd1 "guts.vhd"
analyze -f vhd1 "inside.vhd"
analyze -f vhd1 "test.vhd"
elaborate TOP
current_design TOP
remove_constraint -all
set_port_is_pad "*"
insert_pads
compile
write -format db -hierarchy -output TOP +
    "_compiled.db"

```

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

### **test.ucf (FPGA/Design compiler only)**

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

### **test.ucf (FPGA Express only)**

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

## Glossary

---

This appendix contains definitions and explanations for terms used in the Quick Start Guide for Xilinx Alliance Series vM1.3.

### Definitions

#### **aliases**

Aliases, or signal groups, are useful for probing specific groups of nodes.

#### **attribute**

Attributes are instructions placed on symbols or nets in an FPGA schematic to indicate their placement, implementation, naming, direction, or other properties.

#### **AutoRoute**

AutoRoute automatically routes the objects you specify.

#### **block**

A group consisting of one or more logic functions.

#### **component**

A component is an instantiation or symbol reference from a library of logic elements that can be placed on a schematic.

## **constraint**

Constraints are specifications for the implementation process. There are several categories of constraints: routing, timing, area, mapping, and placement constraints.

Using attributes, you can force the placement of logic (macros) in CLBs, the location of CLBs on the chip, and the maximum delay between flip-flops. PPR does not attempt to change the location of constrained logic.

CLBs are arranged in columns and rows on the FPGA device. The goal is to place logic in columns on the device to attain the best possible placement from the point of view of performance and space.

## **Core Technology Tools**

A set of tools that comprise the mainstream programs offered in the Xilinx design implementation tools. The tools are: NGDBuild, MAP, PAR, NGDAnno, TRCE, all the NGD2 translator tools, BitGen, PROMGen, and EPIC.

## **DC2NCF**

DC2NCF (design constraints to netlist constraints file) translates a Synopsys DC file to a Netlist Constraints File (NCF). The DC file is a Synopsys setup file containing constraints for the design.

## **guided mapping**

An existing NCD file is used to “guide” the current MAP run. The guide file may be used at any stage of implementation: unplaced or placed, unrouted or routed.

## **HDL**

HDL (Hardware Description Language).

## **LCA file**

An LCA file is a mapped file of a Xilinx design produced by an earlier release.

## **LCA2NCD**

LCA2NCD converts an LCA file to an NCD file. The NCD file produced by LCA2NCD can be placed and routed, viewed in EPIC, analyzed for timing, and back-annotated.

## **LogiBLOX**

Xilinx design tool for creating high-level modules such as counters, shift registers, and multiplexers.

## **locking**

Lock placement applies a constraint to all placed components in your design. This option specifies that placed components cannot be unplaced, moved, or deleted.

## **logic**

Logic is one of the three major classes of ICs in most digital electronic systems: microprocessors, memory, and logic is used for data manipulation and control functions that require higher speed than a microprocessor can provide.

## **Logic Block Editor**

The Logic Block Editor allows you to edit the internal logic of a selected programmable component. Use the Edit Block command to start the logic block editor.

## **macro**

A macro is a component made of nets and primitives, flip-flops or latches, that implements high-level functions, such as adders, subtractors, and dividers. Soft macros and RPMs are types of macros.

A macro can be unplaced, partially placed or fully placed, and it can also be unrouted, partially routed, or fully routed. See also “physical macro.”

## **MCS file**

An MCS file is an output from the PROMGen program in Intel's MCS-86 format.

## **MDF file**

An MDF (MAP directive file) file is a file describing how logic was decomposed when the design was originally mapped. The MDF file is used for guided mapping using Xilinx Development System software. The MDF file enables the MAP program to recreate the decompositions chosen when the design was first mapped.

## **MRP file**

An MRP (mapping report) file is an output of the MAP run. It is an ASCII file containing information about the MAP run. The information in this file contains DRC warnings and messages, mapper warnings and messages, design information, schematic attributes, removed logic, expanded logic, signal cross references, symbol cross references, physical design errors and warnings, and a design summary.

## **NCD file**

An NCD (netlist circuit description) file is the output design file from the MAP program, LCA2NCD, PAR, or EPIC. It is a flat physical design database correlated to the physical side of the NGD in order to provide coupling back to the user's original design. The NCD file is an input file to MAP, PAR, TRCE, BitGen, and NGDAnno.

## **NCF file**

An NCF (netlist constraints file) file is produced by a synthesis vendor toolset, or by the DC2NCF program. This file contains constraints specified within the toolset. EDIF2NGD reads the constraints in this file and adds the constraints to the output NGO file.



## **NGDAnno**

The NGDAnno program distributes delays, setup and hold time, and pulse widths found in the physical NCD design file back to the logical NGD file. NGDAnno merges mapping information from the NGM file, and timing information from the NCD file and puts all this data in the NGA file.

## **NGA file**

An NGA (native generic annotated) file is an output from the NGDAnno run. An NGA file is subsequently input to the appropriate NGD2 translation program.

## **NGD2EDIF**

NGD2EDIF is a program that produces an EDIF 2.1.0 netlist in terms of the Xilinx primitive set, allowing you to simulate pre- and post-route designs.

## **NGD2XNF**

NGD2XNF is a program that translates a Xilinx Development System (NGD) format file into a Xilinx netlist (XNF) file.

## **NGD2VER**

NGD2VER is a program that translates your design into a Verilog HDL file containing a netlist description of the design in terms of Xilinx simulation primitives.

## **NGD2VHDL**

NGD2VHDL is a program that translates your design into a Vital 3 compliant VHDL file containing a netlist description of your design in terms of Xilinx simulation primitives.

## **NGDBuild**

The NGDBuild program performs all the steps necessary to read a netlist file in XNF or EDIF format and create an NGD file describing the logical design.

## **NGD file**

An NGD (native generic database) file is an output from the NGDBuild run. An NGD file contains a logical description of the design expressed both in terms of the hierarchy used when the design was first created and in terms of lower-level Xilinx primitives to which the hierarchy resolves.

## **NGM file**

An NGM (native generic mapping) file is an output from the MAP run and contains mapping information for the design. The NGM file is an input file to the NGDAnno program.

## **PAR (Place and Route)**

PAR is a program that takes an NCD file, places and routes the design, and outputs an NCD file. The NCD file produced by PAR can be used as a guide file for reiterative placement and routing. The NCD file can also be used by the bitstream generator, BitGen.

## **path delay**

A path delay is the time it takes for a signal to propagate through a path.

## **PCF file**

The PCF file is an output file of the MAP program. It is an ASCII file containing physical constraints created by the MAP program as well as physical constraints entered by you. You can edit the PCF file from within EPIC.

## **physical Design Rule Check (DRC)**

Physical Design Rule Check (DRC) is a series of tests to discover logical and physical errors in the design. Physical DRC is applied from EPIC, BitGen, PAR, and Hardware Debugger. By default, results of the DRC are written into the current working directory.

## physical macro

A physical macro is a logical function that has been created from components of a specific device family. Physical macros are stored in files with the extension `.nmc`. A physical macro is created when EPIC is in macro mode. See also “macro.”

## pin

A pin can be a symbol pin or a package pin. A package pin is a physical connector on an integrated circuit package that carries signals into and out of an integrated circuit. A symbol pin, also referred to as an instance pin, is the connection point of an instance to a net.

## pinwires

Pinwires are wires which are directly tied to the pin of a site (i.e. CLB, IOB, etc.)

## route

The process of assigning logical nets to physical wire segments in the FPGA that interconnect logic cells.

## route-through

A route that can pass through an occupied or an unoccupied CLB site is called a route-through. You can manually do a route-through in EPIC. Route-throughs provide you with routing resources that would otherwise be unavailable.

## states

The values stored in the memory elements of a device (flip-flops, RAMs, CLB outputs, and IOBs) that represent the state of that device for a particular readback (time). To each state there corresponds a specific set of logical values.

## **TRCE**

TRCE (Timing Reporter and Circuit Evaluator) “trace” is a program that will automatically perform a timing analysis on a design using available timing constraints. The input to TRCE is a mapped NCD file and, optionally, a PCF file. The output from TRCE is an ASCII timing report which indicates how well the timing constraints for your design have been met.

(Historical note: TRCE should not be confused with the UNIX trace command. The UNIX trace command is used to trace system calls and signals).

## **TWR file**

A TWR (Timing Wizard Report) file is an output from the TRCE program. A TWR file contains a logical description of the design expressed both in terms of the hierarchy used when the design was first created and in terms of lower-level Xilinx primitives to which the hierarchy resolves.

## **wire**

A wire is either: 1) a net or 2) a signal.

## **UCF file**

A UCF (user constraints file) contains user-specified logical constraints.

# 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.Xilinx.com](http://www.Xilinx.com), 1-4









The Programmable Logic Company<sup>SM</sup>

2100 Logic Drive  
San Jose, CA 95124-3400  
Tel: 1-408-559-7778  
Fax: 1-408-559-7114  
web: [www.xilinx.com](http://www.xilinx.com)



0401649