Mentor Graphics Interface/ Tutorial Guide

Introduction

Getting Started

Design Techniques

FPGA Design Issues

EPLD Design Issues

Functional Simulation Preparation

Design Implementation

Timing Simulation Preparation

Simulation Issues

Manual Translation

Design Architect Tutorial

QuickSim Tutorial

Mentor Graphics Interface/Tutorial Guide — 0401408 01 *Printed in U.S.A.*

Mentor Graphics Interface/ Tutorial Guide

X-BLOX Tutorial

Xilinx ABEL Tutorial

XACT-Performance and XDelay Tutorial

XEPLD Tutorial

Error Messages

XILINX®, XACT, XC2064, XC3090, XC4005, and XC-DS501 are registered trademarks of Xilinx. All XC-prefix product designations, 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, HardWire, LCA, Logic Cell, LogicProfessor, MicroVia, PLUSASM, SMARTswitch, UIM, VectorMaze, VersaBlock, VersaRing, and ZERO+ are trademarks of Xilinx. The Programmable Logic Company and The Programmable Gate Array Company are service marks of Xilinx.

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. PAL and PALASM are registered trademarks 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, QuickSim, QuickSim II, and EXPAND are trademarks of Mentor Graphics, Inc. Sun is a registered trademark of Sun Microsystems, Inc. SCHEMA II+ and SCHEMA III are trademarks of Omation Corporation. OrCAD is a registered trademark of OrCAD Systems Corporation. Viewlogic, Viewsim, and Viewdraw are registered trademarks of Viewlogic Systems, Inc. CASE Technology is a trademark of CASE Technology, a division of the Teradyne Electronic Design Automation Group. DECstation is a trademark of Digital Equipment Corporation. Synopsys is a registered trademark of Synopsys, Inc. Verilog is a registered trademark of Cadence Design Systems, Inc.

Xilinx 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 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 will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx 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; 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 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 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.

Preface

About This Manual

This manual explains how to use Version 5 of the XACT Mentor Graphics interface software with Mentor Graphics software Version 8.2_5. Included in this book is information on using the Mentor Graphics Design Manager interface configured for the design, simulation, and implementation of Xilinx Programmable Logic Devices (PLDs). The following Mentor Graphics and Xilinx applications are represented in the Design Manager window as icons:

- **Mentor Graphics Applications**
  - **Design Architect**: design application with symbol editor, schematic editor, and VHDL editor
  - **QuickSim II**: interactive logic simulator
  - **QuickPath**: static analysis application
  - **Design Viewpoint Editor**: interactive application that allows you to change the rules that configure your design
  - **Notepad Editor**: full-featured, window-based text editor
- **Xilinx Applications**
  - **FNCSIM8**: prepares your designs for functional simulation
  - **TIMSIM8**: prepares your designs for timing simulation
  - **Men2XNF8**: translates your design into a Xilinx netlist file
  - **XMake**: automatically converts your FPGA designs into LCA and BIT files
  - **XEMake**: automatically converts your EPLD designs into

design database and Intelhex files

- **XACT Design Manager**: menu-driven user interface to the Xilinx core tools

Note: PLDs include FPGA and EPLD devices. When applicable, PLD will be used in this manual to include both types of devices. FPGA or EPLD will be used when a distinction is necessary. FPGAs include the XC2000 device family (XC2000, XC2000L), the XC3000 device family (XC3000, XC3000A, XC3000L, XC3100, XC3100A) and the XC4000 device family (XC4000,XC4000A, XC4000H). EPLDs include the XC7000 device family.

The tutorials in this manual provide step-by-step instructions on creating your PLD designs using Design Architect; simulating your designs with QuickSim II; and incorporating X-BLOX and Xilinx ABEL elements into your designs.

This manual assumes that you are familiar with the Mentor Graphics CAE system, Xilinx core tools, and the UNIX operating system. This manual is not intended to replace Mentor Graphics or Xilinx manuals that describe in detail Design Manager, Design Architect, Xilinx Libraries, and Xilinx core software. Refer to the list of related publications below for additional documentation that may be useful.

Manual Contents

This manual covers the following topics:

- **Chapter 1: Introduction.** Introduction to the Mentor Graphics Design Manager interface, general description of the Xilinx design flow, and a brief list of new features in the current software release.
- **Chapter 2: Getting Started.** Configuring your system for XACT Mentor Graphics interface software, and invoking and using Design Manager and Design Architect.
- **Chapter 3: Design Techniques.** General design techniques for creating PLD schematics, including adding and modifying Xilinx attributes.
- **Chapter 4: FPGA Design Issues.** FPGA- specific design issues, including a description of Xilinx FPGA attributes.

- **Chapter 5: EPLD Design Issues.** EPLD- specific design issues, including a description of Xilinx EPLD attributes.
- **Chapter 6: Functional Simulation Preparation.** Preparing your designs for functional simulation.
- **Chapter 7: Design Implementation.** Converting FPGA design files to LCA or BIT files, and converting EPLD design files to design database or Intel Hex files.
- **Chapter 8: Timing Simulation Preparation.** Preparing your designs for timing simulation.
- **Chapter 9: Simulation Issues.** Issues you need to take into consideration when simulating PLDs.
- **Chapter 10: Manual Translation.** Manually processing your design from the operating system command line.
- **Chapter 11: Design Architect Tutorial.** Steps you through a typical FPGA design procedure from schematic entry to completing a functioning device using Mentor Graphics Design Architect configured for Xilinx designs.
- **Chapter 12: QuickSim Tutorial.** Steps you through both a functional and a timing simulation of an FPGA design using Mentor Graphics QuickSim II.
- **Chapter 13: X-BLOX Tutorial.** Creating designs with X-BLOX elements as well as simulating them.
- **Chapter 14: Xilinx ABEL Tutorial.** Creating designs with Xilinx ABEL elements as well as simulating them.
- **Chapter 15: XACT-Performance and XDelay Tutorial.** Specifying timing requirements for your design.
- **Chapter 16: XEPLD Tutorial.** Steps you through a typical EPLD design procedure including entering your schematic, defining PLD equations, fitting your design, and simulating using QuickSim II.
- **Appendix A: Error Messages.** The appendix lists error messages and possible solutions.

Related Publications

This section provides a list of publications that contain information related to the products described in this manual.

Xilinx Publications

XACT User Guide

XACT Reference Guide

X-BLOX User Guide

Xilinx ABEL User Guide

Xilinx Quick Reference Card — Mentor Graphics Design Architect for FPGA Designs

XEPLD Design Guide

XEPLD Reference Guide

Xilinx Quick Reference Card for EPLD Hardware

Xilinx Quick Reference Card for XEPLD Software

Mentor Graphics Publications

Mentor Graphics System Overview Manual

Design Architect User's Manual

Design Architect Reference Manual

Design Manager User's Manual

Design Manager Reference Manual

QuickSim II User's Manual

QuickPath User's and Reference Manual

Design Viewpoint Editor User's and Reference Manual

Notepad User's and Reference Manual

Conventions

The following conventions are used in this manual's syntactical statements:

`Courier font regular` System messages or program files appear in regular Courier font.

`Courier font bold` Literal commands that you must enter in syntax statements are in bold Courier font.

italic font Variables that you replace in syntax statements are in italic font.

`[ ]` Square brackets denote optional items or parameters. However, in bus specifications, such as bus [7:0], they are required.

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

·
·
· A vertical ellipsis indicates material that has been omitted.

`. . .` A horizontal ellipsis indicates that the preceding can be repeated one or more times.

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

↵ This symbol denotes a carriage return.

Contents

Chapter 1 Introduction

Defining the Design Flow 1-1
Defining the Design Manager Interface 1-3
Tools Window Applications... 1-5
Editor 1-5
QuickPath 1-5
QuickSim II 1-5
PLD_DA.. 1-6
PLD_DVE 1-6
PLD_FNCSIM8.. 1-6
PLD_Men2XNF8 1-6
PLD_TIMSIM8 1-7
PLD_XDM.. 1-7
PLD_XEMake 1-7
PLD_XMake 1-7
What is New in this Release 1-8

Chapter 2 Getting Started

Configuring Your System 2-1
Standard Directory Structure 2-2
Converting V7 Designs to V8.x.. 2-3
Entering the Design Manager Environment.................................. 2-3
Invoking Applications in the Design Manager......................... 2-4
Data-centered Invocation 2-5
Tool-centered Invocation 2-5
Entering Design Architect 2-5
Tool-centered Invocation of Design Architect......................... 2-5
Data-centered Invocation of Design Architect 2-6
Retargeting Your Design to a Different Family 2-6
Retargeting in the Design Manager 2-7
Retargeting from the Command Line...................................... 2-8

Chapter 3 Design Techniques

Naming Conventions 3-1
EDIF2XNF 3-2
Signal Naming Conventions 3-3
Symbol Naming Conventions 3-3
Xilinx Libraries 3-4
Primitives and Macros 3-4
X-BLOX 3-4
Using the XACT Libraries 3-5
Bus Rippers 3-5
Creating a Schematic 3-6
Entering Xilinx Attributes 3-7
Adding Properties 3-7
Modifying Properties 3-8
Merging Design Files from Other Sources 3-8
Creating a Xilinx ABEL Symbol 3-9
Adding a Xilinx ABEL Symbol to Your Schematic 3-9

Chapter 4 FPGA Design Issues

FPGA Properties 4-1
Symbol Properties 4-1
REF 4-1
PINTYPE 4-2
INST 4-2
COMP 4-2
FILE 4-2
BLKNM 4-3
HBLKNM 4-4
MAP 4-4
DOUBLE 4-4
DECODE 4-4
INIT 4-5
EQN 4-5
DEF 4-5
CYMODE 4-5
SCHNM 4-5
LIBVER 4-5
Location Properties 4-6
LOC 4-6
RLOC 4-7

USE_RLOC ... 4-7
U_SET ... 4-7
HU_SET ... 4-8
RLOC_ORIGIN ... 4-8
RLOC_RANGE ... 4-8
Input/Output Properties ... 4-8
INTERNAL ... 4-8
NODELAY ... 4-8
FAST or SLOW ... 4-8
MEDFAST or MEDSLOW ... 4-9
RES or CAP ... 4-9
CMOS or TTL ... 4-9
CLB/IOB Properties ... 4-9
BASE ... 4-9
CONFIG ... 4-10
EQUATE_F AND EQUATE_G ... 4-10
Modifying CLB/IOB Properties ... 4-10
Timing Specification Properties ... 4-10
TSidentifier ... 4-11
TNM ... 4-11
TS Flag ... 4-11
Adding TIMESPEC/TIMEGRP Properties ... 4-11
Adding a TS Flag to a Net ... 4-12
Net Properties ... 4-13
NET ... 4-13
NETFLAG ... 4-13

Chapter 5 EPLD Design Issues

Components ... 5-1
Buffers and Pads ... 5-2
Input and Output Buffer Connections ... 5-2
Output Buffers and 3-State Buffers ... 5-3
On-Chip 3-State Multiplexing ... 5-7
Input Buffers, Clocks, and Global Control Nets ... 5-7
EPLD-Specific Components ... 5-8
Counters ... 5-11
Arithmetic Components ... 5-11
PLD Components ... 5-12
Components Not Supported by Specific Devices ... 5-12
Primitives and Macros ... 5-13
User-Defined Primitives and Macros ... 5-14

Creating a User-defined Primitive 5-14
Creating a User-defined Macro 5-14
Assigning Logical High and Low Values 5-15
Attributes 5-15
Using Attributes 5-16
Global Attributes 5-16
PLD Equation File Name: The PLD Attribute 5-17
Pin Assignment: The LOC Attribute 5-18
Power Setting: The LOWPWR Attribute 5-19
Logic Optimization Attributes 5-19
OPT 5-19
LOGIC_OPT 5-20
MINIMIZE 5-20
UIM_OPT 5-20
FOE_OPT 5-21
CLOCK_OPT 5-21
REG_OPT 5-21
PRELOAD_OPT 5-21
Fast or High-Density Function Blocks: F and H Attributes 5-22
MRINPUT 5-23

Chapter 6 Functional Simulation Preparation

PLD_Men2XNF8 6-1
Design Object 6-3
Part Type 6-3
Run MemGen Only 6-3
Verbose Output 6-3
Help 6-3
OK or Cancel 6-3
PLD_FNCSIM8 6-4
Design Object 6-5
Schematic 6-5
Use Original 6-5
Auto Generate 6-5
Run QuickSim 6-6
Verbose Output 6-6
Help 6-6
OK or Cancel 6-6
Output Files 6-6
QuickSim II 6-7
Design Pathname 6-8

Symbol 6-8
Interface 6-8
Timing Mode 6-8
Simulator Resolution 6-9
OK, Reset, or Cancel 6-9
PLD_DVE 6-9
Design Object 6-10
Create Viewpoint Type 6-10
Simulation 6-10
XNF Translation 6-10
Back Annotation 6-10
PLD Technology 6-10
Use Default Viewpoint Name 6-10
Help 6-10
OK, Reset, or Cancel 6-11

Chapter 7 Design Implementation

PLD_XMake 7-1
Design Object 7-2
Override Part Type 7-2
Verbose Output 7-2
Rerun All Steps 7-3
Use Guide File 7-3
Perform X-BLOX Optimization 7-3
Generate MAK File Only 7-3
Output to Screen 7-3
Mapping Strategy 7-3
Map-Then-Merge 7-4
Map-FILE=-Then-Merge 7-4
Merge-Then-Map 7-4
Target 7-4
Make Bitstream 7-4
Make Placed & Routed Design 7-4
Stop to Review DRC 7-4
File 7-5
OK or Cancel 7-5
PLD_XEMake 7-5
Design Object 7-6
Override Part Type 7-6
Generate MAK File Only 7-6
Force Execution 7-6

Target 7-6
Signature 7-7
Output Files 7-7
Design File 7-7
Programming File 7-7
MAK File 7-8
Report File 7-8
Out File 7-8
PRP File 7-8
Men2XNF8.log File 7-8

Chapter 8 Timing Simulation Preparation

PLD_TIMSIM8 8-1
Design Object 8-2
Schematic 8-2
Use Original 8-3
Auto Generate 8-3
Run QuickSim 8-3
Verbose Output 8-3
Help 8-4
OK or Cancel 8-4
Output Files 8-4
QuickSim II 8-4
Design Pathname 8-5
Symbol 8-5
Interface 8-5
Timing Mode 8-5
Simulator Resolution 8-6
OK, Reset, or Cancel 8-6
QuickPath 8-6

Chapter 9 Simulation Issues

Simulation Models 9-1
Analyzing Nets from the Schematic 9-1
FPGA Devices 9-2
Global Reset and 3-State Signals 9-2
XC4000 Simulation Exceptions 9-3
EPLD Devices 9-3
Using PRLD for Initialization 9-3
XC7000 Simulation Exceptions 9-5

Chapter 10 Manual Translation

Functional Simulation 10-2
FPGA and EPLD Designs with Only Schematic Elements 10-2
FPGA Designs with Schematic and CLB/IOB/EQN Elements 10-4
FPGA Designs with Schematic and X-BLOX Elements 10-6
FPGA Designs with Schematic Elements and Elements with the FILE Property 10-8
FPGA Designs with Schematic Elements, Elements with the FILE Property, and X-BLOX Elements 10-10
Design Implementation 10-11
FPGA Designs 10-11
EPLD Designs 10-13
Timing Simulation 10-14
FPGA Designs with Schematic Elements Only 10-14
FPGA Designs with Non-schematic Elements 10-16
EPLD Designs 10-18
Program Summary 10-18
EDIF2XNF 10-18
Syntax 10-19
Variables 10-19
Options 10-19
ENWRITE 10-22
Syntax 10-22
Variables 10-22
Options 10-22
FNCSIM8 10-22
Syntax 10-23
Variables 10-23
Options 10-23
Gen_Sch8 10-23
Syntax 10-23
Variables 10-23
Options 10-24
LCA2XNF 10-24
Men2XNF8 10-24
Syntax 10-24
Variables 10-24
Options 10-25
PLD_DVE 10-25

Syntax 10-25
Variables 10-25
Options 10-25
PLD_DVE_BA 10-26
Syntax 10-26
Variables 10-26
Options 10-26
PLD_DVE_SIM 10-26
Syntax 10-26
Variables 10-26
Options 10-27
QUICKSIM II 10-27
Syntax 10-27
Variables 10-27
Options 10-27
TIMSIM8 10-27
Syntax 10-27
Variables 10-28
UNAKAXNF 10-28
Syntax 10-28
Variables 10-28
Options 10-28
VMH2XNF 10-29
X-BLOX 10-29
Syntax 10-29
Variables 10-29
XBLXGS 10-30
Syntax 10-30
Variables 10-30
Options 10-31
XDelay 10-31
XNFBA 10-31
Syntax 10-31
Variables 10-31
Options 10-32
Examples 10-32
XNFMerge 10-34
XNFPrep 10-35

Chapter 11 Design Architect Tutorial

Required Background Knowledge 11-2

FPGA Design Flow 11-2
Software Installation.. 11-5
    Required Software 11-5
    Before Beginning the Tutorial 11-5
        Modifying Mentor Graphics Variables.. 11-6
    Installing the Tutorial... 11-7
        Standard Directory Structure 11-8
        Tutorial Directory and Files... 11-8
Starting the Design Manager 11-11
    Tools Window 11-12
    Navigator Window.. 11-13
    Command Palette 11-13
Copying the Tutorial Files 11-13
Targeting the Design for the XC4000 Family...................................... 11-14
Starting Design Architect 11-15
    Using the Mouse in Design Architect... 11-16
        Left Mouse Button 11-16
        Middle Mouse Button (Strokes) 11-16
        Right Mouse Button 11-17
    Using the Function Keys... 11-17
    Selecting Commands from the Menu Bar....................................... 11-18
    Selecting Commands from the Palette 11-18
    Entering Commands from the Keyboard 11-18
    Cancelling Commands.. 11-18
    Repeating Menu Commands 11-18
    Manipulating the Screen 11-19
Completing the Calc Design 11-19
    Design Description... 11-19
    Creating the ANDBLK2 Symbol.. 11-21
        Opening a Symbol Window 11-21
        Creating the Symbol Outline.. 11-21
        Adding Pins to the ANDBLK2 Symbol 11-22
        Adding Text 11-25
        Modifying Text Size 11-26
        Saving the ANDBLK2 Symbol 11-27
    Creating the ORBLK2 Symbol 11-28
    Creating Schematics for ANDBLK2 Symbol................................... 11-30
        Opening a Schematic Window 11-30
        Adding the First Component to a Schematic 11-30
        Placing Additional Components.. 11-32
        Copying a Component... 11-33

Moving a Component.. 11-34
Adding Buses to a Schematic... 11-35
Adding Nets to a Schematic 11-36
Completing the Net Connections 11-38
Labeling Nets and Buses.. 11-39
Adding Ports 11-42
Labeling Buses 11-44
Defining Bus Ripper Rule Properties 11-45
Saving the Schematic 11-47
Creating Schematics for ORBLK2 Symbol 11-48
Completing the ALU Schematic 11-50
Placing User-Created Components 11-51
Placing Library Components.. 11-53
Adding Nets, Buses, Ports and Labels 11-54
FD4CE 11-54
ANDBLK2 and ORBLK2 11-55
Adding Labels to Components... 11-56
Saving the ALU Schematic 11-58
Exploring Xilinx Library Elements 11-58
Viewing a Xilinx Soft Macro Schematic.................................. 11-59
Viewing a Xilinx Library Primitive 11-60
Viewing a Xilinx RPM (XC4000 Family Only) 11-61
Opening the Calc Schematic 11-63
Using the XC3000 Oscillator... 11-64
Using the XC4000 Oscillator... 11-65
Inverting Output Display Signals.. 11-67
Controlling FPGA Layout from the Schematic 11-69
Assigning Pin Locations... 11-69
Designating FAST Pads... 11-71
Using the I/O Flip-Flops 11-72
Saving the Calc Schematic 11-73
Optimizing the Design for the XC4000 Family 11-73
Device-Independent Stack Implementation 11-74
RAM Stack Implementation (XC4000 Family Only) 11-75
Device-Independent State Machine....................................... 11-77
Wide-Edge Decoders (XC4000 Family Only) 11-78
Configuring XMake using XDM 11-80
Using a Constraints File... 11-81
Using .PRO Files 11-83
Using PLD_Men2XNF8.. 11-83
Examining PLD_Men2XNF8 Output Files.............................. 11-84

Using PLD_XMake 11-85
- Examining PLD_XMake Output Files 11-87
- Checking for Warnings in the OUT and PRP Files 11-88
- Checking the RPT File 11-91

Examining Routed Designs with XDE 11-93
- Entering the Design Editor 11-93
- Finding a Block 11-95
- Highlighting a Net 11-96
- Using Command Line Entry 11-96
- Running the Design Rule Checker 11-96

Verifying the Design Using a Demonstration Board 11-97
- Connecting the Cable for Download 11-97
  - XC3000/XC4000 Demonstration Board 11-99
  - XC4000 Demonstration Board 11-100
  - XC3000 Demonstration Board 11-101
- Downloading the Bitstream 11-101
- Testing the Design 11-103

Making Incremental Design Changes 11-105
- Creating the Guide LCA File 11-106
- Making an Incremental Schematic Change 11-106
- Translating the Incremental Design 11-108
- Checking for Errors in the calc.out File 11-109
- Verifying the Change in the Demonstration Board 11-109

Leaving XDM 11-110

Command Summaries 11-111
- Basic Translation for XC3000A and XC3000L Designs 11-111
- Basic Translation for XC4000 Family Designs 11-111
- Basic Translation for XC3000, XC3100, and XC2000 Family Designs 11-112
- Incremental Translation for XC3000A and XC3000L Designs 11-112
- Incremental Translation for XC4000 Family Designs 11-112
- Incremental Translation for XC3000, XC3100, and XC2000 Family Designs 11-113
- Further Reading 11-113

Chapter 12 QuickSim Tutorial

Required Background Knowledge 12-1

Software Installation 12-2
- Required Software 12-2
- Before Beginning the Tutorial 12-2

Modifying Mentor Graphics Variables 12-3
Installing the Tutorial 12-4
Starting PLD_DMGR 12-5
Making a Working Copy of the CALC design 12-5
Basic Functional Simulation 12-6
Preparing the Calc Schematic for Simulation 12-6
PLD_Men2XNF8 12-7
PLD_FNCSIM8 12-7
Viewing the Calc Schematic 12-7
Selecting Nets for Simulation 12-8
Opening Trace and List Windows 12-10
Adding Traces Manually 12-12
Assigning Values to the Clock 12-13
Asserting Global Reset (XC2000 & XC3000 Families Only) 12-15
Asserting Global Set Reset (XC4000 Family Only) 12-16
Design Description 12-17
Simulating the Circuit 12-18
Saving the Results 12-24
Using the Transcript 12-26
Performing a Timing Simulation with PLD_TIMSIM8 12-27
Using PLD_TIMSIM8 to Prepare for Timing Simulation 12-27
Examining the timsim8.log File 12-28
Simulating with a Command File in QuickSim 12-28
Timing Simulation Command Summary 12-29

Chapter 13 X-BLOX Tutorial

Introduction 13-1
Before Beginning the Tutorial 13-1
Required Software 13-1
Preparing the Design 13-2
Modifying the Calc Design 13-2
Adding X-BLOX Module to Calc 13-3
Viewing the ALU_BLOX Schematic 13-3
Completing the ALU_BLOX Schematic 13-5
Understanding X-BLOX Buses 13-5
Using BUS_DEF Symbols to Define Bus Widths 13-6
Completing the Bus Definition 13-7
X-BLOX Symbols 13-8
Functional Simulation 13-10
Using PLD_Men2XNF8 13-11

Men2XNF8 Log File 13-11
Using PLD_FNCSIM8 13-13
FNCSIM8 Output 13-13
Viewing the Simulation Schematic 13-15
Using QuickSim II 13-17
Implementing the Calc Design 13-18
PLD_XMake Output Window 13-18
Verifying Calc on the Demonstration Board 13-20
Timing Simulation 13-21
PLD_TIMSIM8 Output Window 13-21
Using QuickSim II 13-23
Further Reading 13-24

Chapter 14 Xilinx ABEL Tutorial

Introduction 14-1
Before Beginning the Tutorial 14-1
Required Software 14-1
Preparing the Design 14-2
Copying the Tutorial Files 14-3
Modifying the Calc Design 14-3
Viewing STAT_ABL.ABL 14-3
Xilinx ABEL Output 14-7
Verifying STAT_ABL 14-11
Synthesizing STAT_ABL.ABL 14-11
Creating a Symbol for STAT_ABL 14-12
Adding STAT_ABL to Calc 14-13
The STAT_ABL symbol 14-14
The FILE Property 14-14
Functional Simulation 14-15
PLD_Men2XNF8 14-16
PLD_Men2XNF8 Output 14-16
PLD_FNCSIM8 14-17
PLD_FNCSIM8 Output 14-18
Viewing the Simulation Schematic 14-18
QuickSim II 14-20
Implementing the Calc Design 14-20
PLD_XMake 14-21
PLD_XMake Output 14-22
Verifying Calc on the Demonstration Board 14-23
Timing Simulation 14-23
PLD_TIMSIM8 14-23

PLD_TIMSIM8 Output 14-24
QuickSim II.. 14-25
Further Reading 14-25

Chapter 15 XACT-Performance and XDelay Tutorial

Before Beginning the Tutorial.. 15-2
Required Software 15-2
Preparing the Design 15-2
Understanding XACT-Performance 15-3
Grouping Symbols with TNM Attributes 15-3
TNMs on Logic Primitives 15-3
TNMs on Higher-Level Macro Symbols 15-4
TNMs on Nets to Tag Flip-Flops 15-4
Grouping Symbols by Predefined Class 15-5
Simplifying Symbol Grouping 15-5
Combining Classes: TIMEGRP Symbol 15-5
Joining Two or More Classes into One 15-6
Using the EXCEPT Statement 15-6
Triggering on RISING or FALLING Clock Edges 15-6
Forming Classes by Output Net Name 15-7
Attaching Timing Specifications: TIMESPEC Symbol 15-7
Deciding When to Use XACT-Performance 15-8
Setting Default Timing Requirements 15-9
Adding a TNM Property 15-10
Entering Default Timing Specifications 15-11
Adding Timing Constraints to Specific Paths 15-13
Defining TNM Groups 15-14
Defining the INFFS Class 15-14
Defining the STACKER Class (XC4000 Family Only) 15-14
Defining the STACKER Class (XC3000A Only) 15-15
Defining the ALUFF Class 15-15
Defining the CTLFF Class 15-16
Defining the STFF Class... 15-16
Grouping Classes with TIMEGRP... 15-17
Specifying TIMESPEC Constraints... 15-19
Making a Final Check 15-20
Implementing the Calc Design 15-22
Creating a Routed Design.. 15-23
Examining XMake Output 15-23
Examining the PPR Log File 15-24
Warnings in the PPR Log File... 15-26

Timing Analysis Summary 15-27
Using XDelay, the Timing Analysis Program 15-27
Analyzing the Calc Design 15-29
Invoking XDelay 15-29
Using the Flagblk Option 15-29
Disabling Paths Through SD/RD Pins of Flip-Flops 15-30
Displaying Current Options 15-30
Using Analyze Mode 15-31
Examining Analyze Mode Output 15-31
Using XDelay-TimeSpec Mode 15-32
Examining XDelay-TimeSpec Mode Output 15-33
Using XDelay Mode 15-38
Reporting by Path Type 15-38
Specifying Source and Destination 15-40
Further Reading 15-42

Chapter 16 XEPLD Tutorial

Software Installation 16-1
Required Software 16-1
Before Beginning the Tutorial 16-1
Modifying Mentor Graphics Variables 16-2
How to Follow this Tutorial 16-3
The Tutorial Design 16-5
The Example Files 16-6
Mentor XEPLD Demonstration Procedure 16-7
Overview of the Sessions 16-9
Session 1: Using the XEPLD Software 16-10
Step 1: Prepare the System 16-10
Step 2: Start the Mentor Graphics Design Manager 16-11
Step 3: Select Menu Items 16-11
Session 2: Drawing the Design in Design Architect 16-12
Step 1: Open and View the Existing Design 16-13
Step 2: Create a New Schematic 16-15
Step 3: Change Zoom Level 16-15
Step 4: Enter and Arrange Symbols 16-16
Step 5: Create a Bus 16-18
Step 6: Create Wires 16-20
Step 7: Add Names 16-22
Step 8: Add A Bus Name 16-25
Step 9: Assign Xilinx EPLD Attributes 16-28
Step 10: Finish the Drawing 16-30

Step 11: Assign Signals to Specific EPLD Pins 16-31
Step 12: Check Your Design.. 16-31
Step 13: Save Your Design.. 16-32
Step 14: Exit Design Architect 16-32
Session 3: Defining the PLD Equations 16-32
Step 1: Define the Declaration Statements............................ 16-33
Step 2: Create the Boolean Equations................................... 16-33
Session 4: Fitting the Design 16-37
Step 1: Invoke the Fitter.. 16-37
Alternative Ways to Process PLDs.................................. 16-37
Step 2: View the Reports 16-38
Session 5: Simulating the Design 16-39
Step 1: Prepare Your Input Vectors....................................... 16-40
Step 2: Run the Simulation and View the Results 16-42
Session 6: Completely Schematic-Based Designs and Functional Simulation.. 16-45
Step 1: Change the Working Directory 16-45
Step 2: Create a Custom Symbol 16-45
Open a Symbol Window 16-45
Draw the Rectangle 16-46
Add the Pins 16-46
Lengthening the Pins.. 16-48
Add the Symbol Name.. 16-49
Check the Symbol... 16-49
Save the Symbol... 16-49
Step 3: Place the Custom Symbol in the Schematic............... 16-50
Step 4: Create the Lower-Level Schematic 16-52
Step 5: Run the Simulation and View the Results 16-54
Step 6: Quit Mentor Graphics 16-55

Appendix A Error Messages

EDIF2XNF.. A-1
Gen_Sch8 .... A-4
Gen_Sym8 .... A-5
PLD_DVE_BA.. A-6
PLD_DVE... A-7
PLD_DVE_SIM A-7
UNAKAXNF.. A-7
XBLXGS... A-9
XNFBA A-10

Mentor Graphics Interface/ Tutorial Guide

Introduction

Printed in U.S.A.

Chapter 1

Introduction

This chapter provides a general introduction to the Mentor Graphics Design Manager application configured for Xilinx designs. The Design Manager is a Motif-compliant, easy-to-use graphic interface that represents applications and design files as icons. You can use the Design Manager to run Xilinx and Mentor Graphics applications and to manage your design files. The Mentor Graphics schematic capture program, Design Architect, is represented in the Design Manager by the PLD_DA icon. PLD_DA is the Design Architect program configured for Xilinx designs. You can use PLD_DA to create your Programmable Logic Device (PLD) designs.

It is highly recommended that you perform the tutorials provided in this manual to become familiar with the basic concepts of PLD design, verification, and implementation.

Defining the Design Flow

The Xilinx design flow is an iterative process of entering, implementing, and verifying PLD designs until they are correct and complete. This manual describes each step in the Design Flow and how to process a design from entering a schematic to generating the appropriate file for downloading to a Xilinx device. There are two ways to perform the necessary steps in the PLD design flow. The easiest and most automatic way is to use the application icons in the Design Manager window. You can also run the various programs in the design flow manually from the shell-level environment of the workstation as described in a subsequent chapter in this manual.

The following is a summary of the steps you need to follow when designing PLDs:

1. Enter your design with the Design Architect schematic editor, making sure that you observe the Xilinx design requirements noted in this manual.
2. Test the functionality of your design. Generate a simulation viewpoint with PLD_FNCSIM8 and then perform simulation with QuickSim II.

   When the functional verification test indicates that your design is working correctly, proceed to the next step, design implementation.
3. Implement your PLD design. Generate a Logic Cell Array (LCA) or VMH / VMD file automatically by invoking PLD_XMake for FPGAs or PLD_XEMake for EPLDs.
4. Test the timing of your design. Use PLD_TIMSIM8 to back-annotate timing information to your original design or to create a new schematic for timing simulation.
5. Download your FPGA design to the appropriate device using the XChecker or MakePROM program. For EPLD designs, copy the .prg file generated by XEMake to a PC and use Prolink to download to the device programmer.

The following diagram provides an overview of this Design Flow using the Design Manager icons.

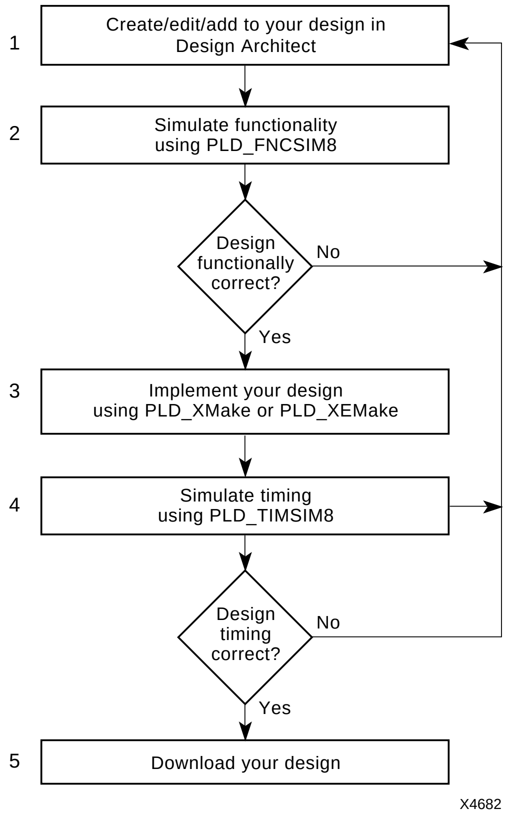

1
Create/edit/add to your design in Design Architect
2
Simulate functionality using PLD_FNCSIM8
Design functionally correct?
No
Yes
3
Implement your design using PLD_XMake or PLD_XEMake
4
Simulate timing using PLD_TIMSIM8
Design timing correct?
No
Yes
5
Download your design
X4682

Figure 1-1 Xilinx Design Flow

Defining the Design Manager Interface

The Mentor Graphics Design Manager is a Motif-compliant, easy-to-use interface that represents applications and design files as icons. You can now perform many tasks in the Design Manager window that were previously done at the operating system level. The Design Manager runs in a window on your workstation display and makes it easy for you to invoke applications and to manage designs, files, and directories. The Design Manager lets you do these tasks by using graphical “point-and-click” actions. You can run applications by selecting an application icon or a design object icon.

Note: A design object consists of the files and directories that make up your design.

The Xilinx script, PLD_DMGR, configures the Design Manager for the creation, implementation, and simulation of Xilinx designs. This manual describes only the Xilinx-configured Design Manager; refer to Mentor Graphics documentation for a more comprehensive description of the interface.

The Design Manager includes a Tools window, a Navigator window, and a Palette as shown in the following figure:

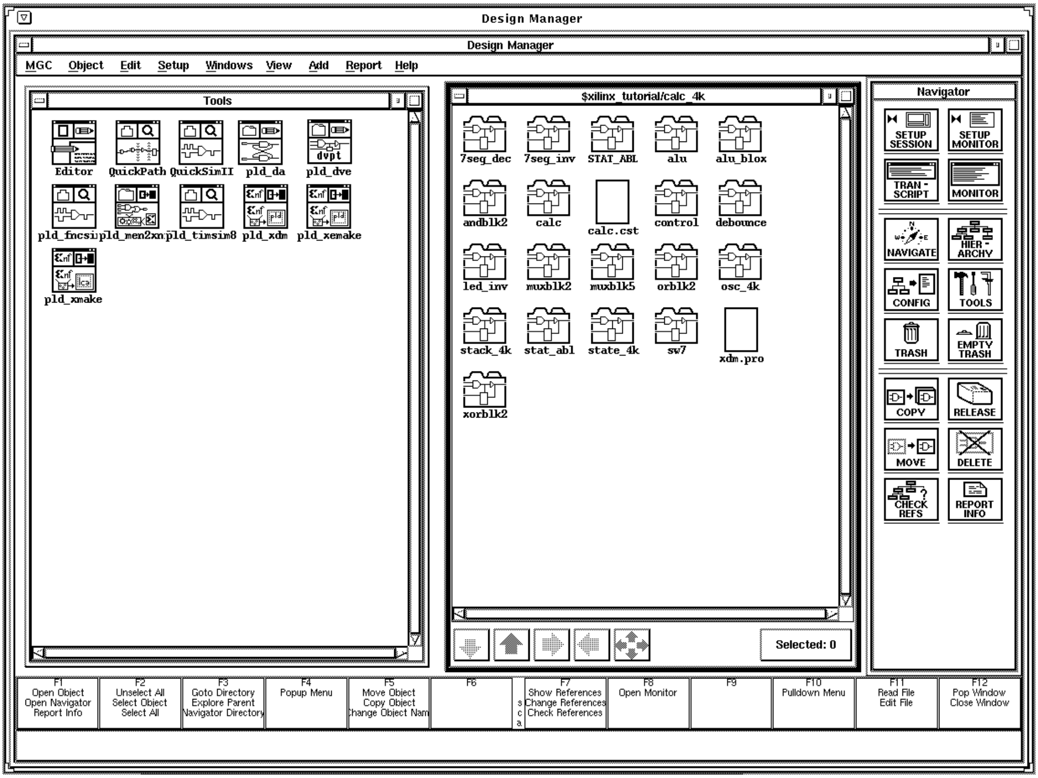


Figure 1-2 Design Manager Window

The Tools window contains icons representing all the Mentor Graphics and Xilinx applications you need to execute the steps in the design flow. The Navigator window contains design object icons, including original schematics as well as files created during translation and simulation. This window makes it easy to access files in different directories. The Palette provides easy access to the most commonly used Design Manager menu items. A brief description of each icon in the Tool window is provided in the following pages.

Tools Window Applications

The Tools Window contains the following applications.

Editor

The Editor icon represents Mentor Graphics Notepad editor. Notepad is a full-featured, window-based text editor. To invoke Notepad, double-click on the Editor icon in the Design Manager window. For more information on Notepad, refer to the Mentor Graphics *Notepad User's and Reference Manual*.

QuickPath

Use the Mentor Graphics QuickPath tool to perform static and slack timing analysis on designs that have been prepared for timing simulation. Refer to the "Timing Simulation Preparation", Simulation Issues", and the tutorial chapters in this manual for more information on QuickPath. For a detailed description of QuickPath, refer to the Mentor Graphics *QuickPath User's and Reference Manual.*

QuickSim II

Use the Mentor Graphics QuickSim II application to perform functional or timing simulation on designs that have been prepared for simulation. QuickSim II is an interactive logic simulator that allows you to verify the functionality of your designs. Refer to the "Functional Simulation Preparation", "Timing Simulation Preparation", "Simulation Issues", and the tutorial chapters in this manual for more information on QuickSim II. For a detailed description of QuickSim II, refer to the Mentor Graphics *QuickSim II User's Manual.*

PLD_DA

Use PLD_DA to execute the script that invokes the Mentor Graphics Design Architect schematic editor configured for Xilinx designs. The Xilinx library of primitives and macros have been added to Design Architect. The Xilinx-configured Design Architect is identical to the Mentor Graphics version except for the addition of these libraries. All the editing commands will function as specified in the Mentor Graphics Design Architect documentation. Refer to the "Design Techniques", "FPGA Design Issues", "EPLD Design Issues", and the tutorial chapters in this manual for more information on creating Xilinx designs with Design Architect. For a more detailed description of Design Architect commands and processes, refer to the Mentor Graphics *Design Architect User's Manual*.

PLD_DVE

Use PLD_DVE to execute the script that invokes the Mentor Graphics Design Viewpoint Editor(DVE) configured for Xilinx designs. When you invoke this application, a dialog box will appear and you will be asked to create either a Simulation, XNF, or Back-Annotation Viewpoint. Refer to the "Functional Simulation Preparation" chapter in this manual for more information on PLD_DVE. For detailed information on DVE, refer to the Mentor Graphics *Design Viewpoint Editor User's and Reference Manual.*

PLD_FNCSIM8

Use PLD_FNCSIM8 to prepare your design for functional simulation. Refer to the "Functional Simulation Preparation", "Simulation Issues", and the tutorial chapters for more information on PLD_FNCSIM8.

PLD_Men2XNF8

Use PLD_Men2XNF8 to translate your design to an XNF file. Refer to the "Functional Simulation Preparation" and the tutorial chapters for more information on PLD_Men2XNF8.

PLD_TIMSIM8

Use PLD_TIMSIM8 to prepare your design for timing simulation. Refer to the “Timing Simulation Preparation”, “Simulation Issues”, and the tutorial chapters for more information on PLD_TIMSIM8.

PLD_XDM

Note: There is no support for Mentor-specific programs in the Xilinx Design Manager (XDM).

Use PLD_XDM to invoke the Xilinx Design Manager (XDM). This application provides a menu-driven user interface to the Xilinx core tools.

XDM gives you the ability to configure the individual programs that are executed by the implementation tools, XMake and XEMake. This capability is helpful when you want to debug a design.

You can also access the XACT Design Editor (XDE) within XDM. XDE allows you to hand-edit a routed design, insert probes while doing in-circuit verification, and perform static timing analysis. XDE cannot be used on EPLD designs.

For more information on XDM, refer to the tutorial chapters of this manual and to the *XACT Reference Guide*.

PLD_XEMake

Use PLD_XEMake to execute the XEMake program on EPLD designs. This tool allows you to execute XEMake without having to enter the Xilinx Design Manager. The XEMake program automatically fits your design into a selected EPLD device and generates a device program file. Refer to the “Design Implementation” chapter of this manual and to the *XEPLD Reference Guide* for more information.

PLD_XMake

Use PLD_XMake to execute the XMake application on FPGA designs. This tool allows you to execute XMake without having to enter the Xilinx Design Manager. The XMake program runs the Xilinx core tools to create a routed LCA file. Refer to the “Design Implementation” chapter of this manual and to the *XACT Reference Guide* for more information on XMake.

What is New in this Release

For a detailed description of new features included in this software release, refer to the Release Notes that came with your Xilinx software package.

- Added Unified Libraries to the XACT Libraries menu in Design Architect. The Unified Libraries use the same component names, with the same footprints (shapes, pin names, and functionality) between product families allowing you to convert designs from one product family to another. Pre-Unified Libraries are categorized as Obsolete and are provided for backward compatibility.
- Added the capability to create, simulate, and implement EPLD designs.
- Added Hard Macro Replacement Library to overwrite old hard macro library. The new library will contain the old hard macro symbols with Relationally Placed Macro (RPM) schematics. Unlike the old hard macros, the replacements will simulate directly.
- Added Mentor Graphics Design Manager support. Xilinx and Mentor Graphics applications are represented as icons in the Design Manager window.
- Added guide feature for XC3000A, XC3000L, XC3100A, and XC4000 designs. The guide feature allows incremental and iterative design techniques previously not supported.
- Revised PPR to place and route XC3000A, XC3000L, and XC3100A designs.
- Revised X-BLOX to process XC3000A, XC3000L, and XC3100A designs.
- Added EPLD information to Mentor Graphics Interface User Guide.
- IPAD, OPAD, IOPAD,UPAD symbols are now Class P.
- Added Design Architect, QuickSim II, X-BLOX, Xilinx ABEL, XACT Performance / XDelay, and XEPLD tutorials to the Mentor Graphics Interface User Guide.

Mentor Graphics Interface/ Tutorial Guide

Getting Started

Chapter 2

Getting Started

This chapter describes how to configure your system to use the Mentor Graphics Design Manager for creating and processing Xilinx designs. A brief description of how to enter the Design Manager and Design Architect is also provided. A procedure for retargeting your designs from one device family to another is also included in this chapter.

Configuring Your System

Install the appropriate software and verify that your system is properly configured as described in the Release Notes that came with your software package. When you have finished the installation, verify that your .cshrc or setup file contains lines similar to the following:

Note: Directory path names will vary.

```
setenv LCA /location_of_ds_344:
setenv XACT /location_of_ds_344:
/location_of_ds502 or location_of_ds550
set PATH=($PATH \
    $LCA/com/sparc\
    $LCA/bin/sparc\
   /location_of_ds502 or location_of_ds550 /bin/sparc\
    )
```

In addition to these Xilinx-specific environment variables, the Mentor Graphics variables MGC_HOME, MGC_GENLIB, LD_LIBRARY_PATH, MGC_LOCATION_MAP, and MGC_WD should be set as follows:

- MGC_HOME: this should point to the Mentor Graphics software tree.

- MGC_GENLIB: this should point to the Mentor Graphics gen_lib library.
- LD_LIBRARY_PATH: This variable is used by the Mentor Graphics Design DataPort (DDP) routines that are accessed by some Xilinx programs. If you are using a Sparc station with OpenWindows installed in /usr, set this variable as follows:

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

- MGC_LOCATION_MAP: this variable should point to a valid location map file. Each component in a design contains a reference indicating where it resides on the disk. All components in designs created in the Mentor Graphics V8.2_5 environment reference the variable $LCA. $LCA must be defined in the file pointed to by $MGC_LOCATION_MAP. See the Mentor Graphics documentation for more information on location maps.
- MGC_WD: this variable should point to the working directory.

Note: Problems occur in the Xilinx scripts if MGC_WD is not set correctly.

Refer to the Release Notes for additional information on paths and environment variables.

Standard Directory Structure

When you create a design in Design Architect, a directory is created in the project directory with the same name as the design. This design directory contains a schematic directory, symbol files, viewpoint files, and part interfaces. It is identified as a design object by the file *design_nam*e.mgc_component.attr.

For example, if you create a schematic named calc, a directory named calc will be created, and, at the same level, the file calc.mgc_component.attr will identify the directory as a design object. The calc directory will contain all the files that describe the calc design such as symbol files and schematic files.

Converting V7 Designs to V8.x

To convert a V7 design to V8.x and for more information about the V8 environment, refer to the *Transition Guide for V8.x Design Capture Products* from Mentor Graphics. In V7, all Xilinx components reference /xact. In V8.2_5, all components reference $LCA. You must update your references after you convert a design from the V7 to V8.2_5 environment. To convert a V8 design to V8.2_5, simply open and save it in V8.2_5 Design Architect.

Entering the Design Manager Environment

To enter the Design Manager from the operating system, type:

```
> pld_dmgr ↵
```

The Design Manager window appears as shown in the following figure:

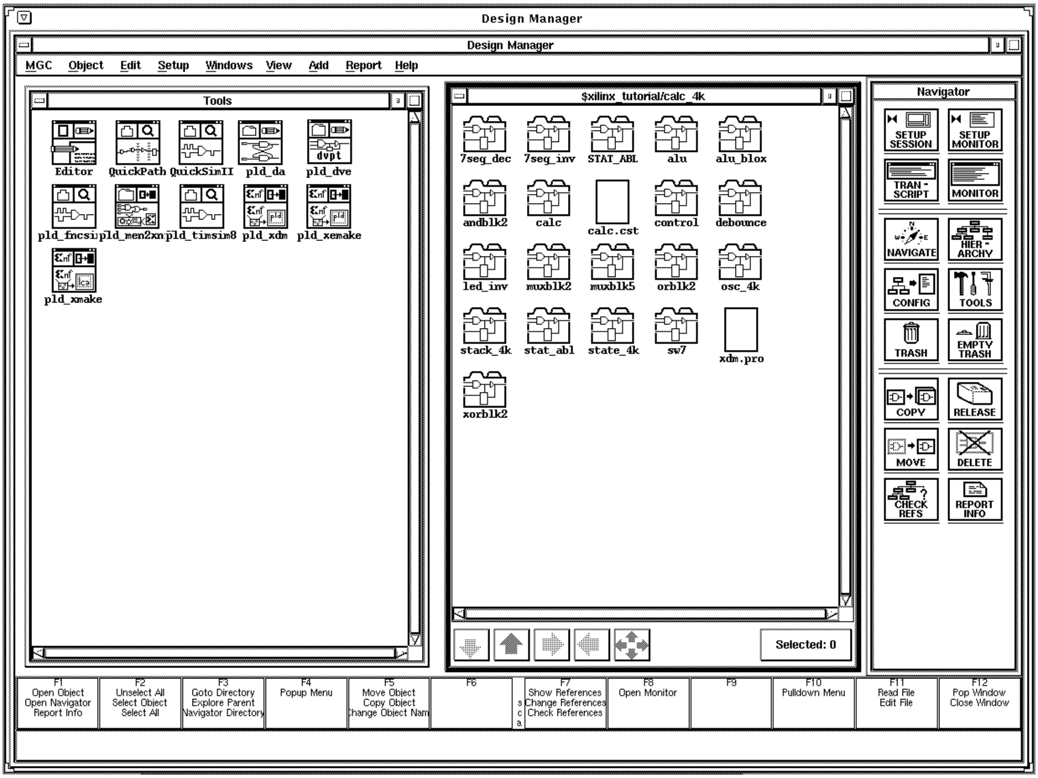


Figure 2-1 Design Manager Window

Invoking Applications in the Design Manager

You can invoke an application in the Design Manager by either performing a data-centered invocation or a tool-centered invocation. The two methods are described below. Refer to Mentor Graphics *Design Manager User’s Manual* for a detailed description of Design Manager operation.

Data-centered Invocation

To perform a data-centered invocation, perform the following steps:

1. Select a design object in the Navigator window and press the right mouse button.
2. Select **Open** from the Navigator menu.
3. Select the appropriate application from the pop-up menu.

Note: Only the applications that can be executed on the selected object will be displayed in the pop-up menu.

4. A dialog box appears with option fields or the application is executed. The dialog box options are described in detail in subsequent sections of this manual.

Tool-centered Invocation

To perform a tool-centered invocation, perform the following steps:

1. Select the appropriate application in the Tools window.
2. Double-click the left mouse button.
3. A dialog box appears with option fields or the application is executed. The dialog box options are described in detail in subsequent sections of this manual.

Entering Design Architect

The Design Architect editing commands should function as specified in the Mentor Graphics *Design Architect User's Manual*. To begin your schematic, enter Design Architect by using one of the following methods:

Tool-centered Invocation of Design Architect

1. Select the PLD_DA icon.
2. Double-click the left mouse button.
3. The Design Architect window appears similar to what is shown in the figure below, but without a schematic displayed. You can use the Open Sheet icon in the session palette to open a schematic sheet.

Data-centered Invocation of Design Architect

1. Select a design in the Navigator window and press the right mouse button.
2. Select **Open** from the Navigator menu.
3. Select **pld_da**.
4. The Design Architect window appears similar to what is shown in the figure below.

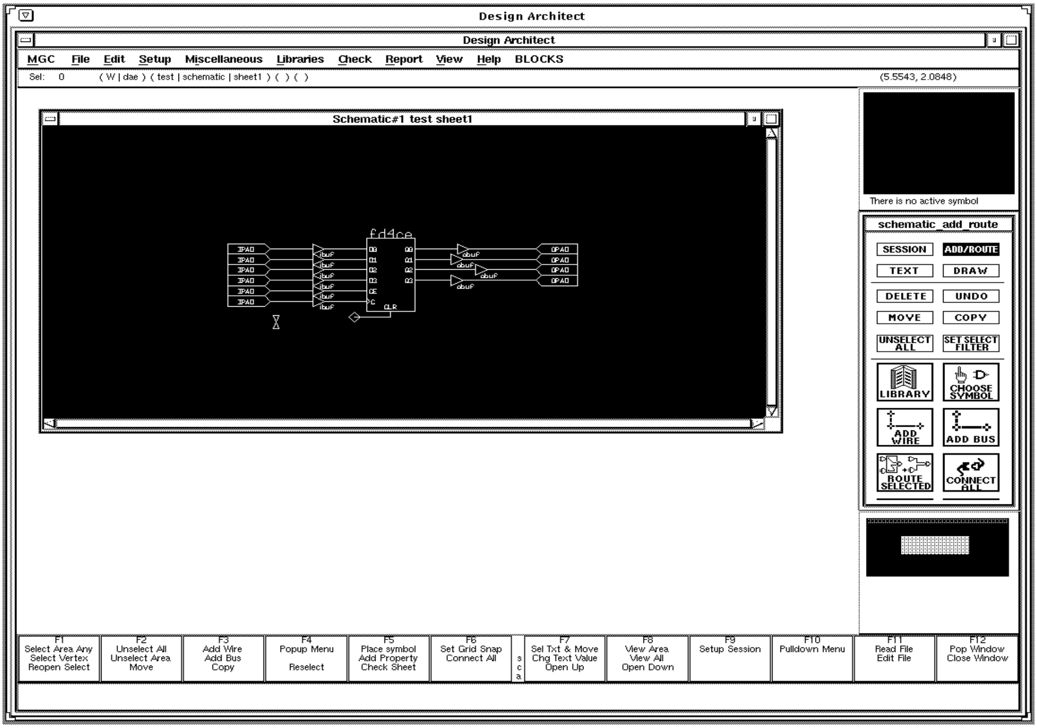

Design Architect
Design Architect
MGC File Edit Setup Miscellaneous Libraries Check Report View Help BLOCKS
Sel: 0 (W | dae) (test | schematic | sheet1) ( ) ( )
(5.5543, 2.0848)
Schematic#1 test sheet1
fd4ce
IPAD
ibuf
obuf
OPAD
CLR
There is no active symbol
schematic_add_route
SESSION
ADD/ROUTE
TEXT
DRAW
DELETE
UNDO
MOVE
COPY
UNSELECT ALL
SET SELECT FILTER
LIBRARY
CHOOSE SYMBOL
ADD WIRE
ADD BUS
ROUTE SELECTED
CONNECT ALL
F1 Select Area Any Select Vertex Reopen Select
F2 Unselect All Unselect Area Move
F3 Add Wire Add Bus Copy
F4 Popup Menu Reselect
F5 Place symbol Add Property Check Sheet
F6 Set Grid Snap Connect All
s c a
F7 Sel Txt & Move Chg Text Value Open Up
F8 View Area View All Open Down
F9 Setup Session
F10 Pulldown Menu
F11 Read File Edit File
F12 Pop Window Close Window

Figure 2-2 Design Architect Window

Retargeting Your Design to a Different Family

The Unified Libraries allow you to retarget your designs from one device family to another provided both your source and target designs only include symbols from the Unified Libraries. Since most of the symbols in the Unified Libraries have the same footprint and name from one device family to another, you can easily convert your designs.

The procedures provided below allow you to change every reference of every design object in your design directory from the source design library to the target design library. In your target design, the symbols that are common to the source and target families maintain their relative location and pin position in the schematic. Pins on these symbols retain their connectivity to the nets they were attached to in the source design.

You must manually replace symbols that are not common to your source and target families with equivalent logic. For example, if a GCLK was used in an XC3000 design that is retargeted for use in an XC4000 device, the GCLK symbol must be manually replaced with a BUFGP, BUFG, or BUFGS, the XC4000 equivalent of a GCLK.

You can either change references within the Design Manager or from the UNIX command line. Both procedures are provided below.

Note: In the following procedures, XC3000 is used as the source design device family and XC4000 is used as the target design device family. You can also retarget other device families.

Retargeting in the Design Manager

1. Select **`MGC`** → **`Location Map`** → **`Set Working Directory`**. A small dialog box appears at the bottom of the screen.
2. Enter the name of the directory above your source design directory in the Directory field of the dialog box. Select **`OK`** or press return. This sets the working directory to the directory above your design directory, and enables you to globally change references in the design directory.
3. Select your design directory in the navigator window.
4. Select **`Edit`** → **`Change`** → **`References`**.
5. A dialog box appears. In the From field, enter the source design device family, such as xc3000. In the To field, enter the target design device family, such as xc4000. Select **`OK`** or press the return key.

6. Check the changes to the path references to the libraries.
7. Open your design in Design Architect and perform a check sheet on each schematic sheet. Correct any errors that might have occurred during the retargeting.

Retargeting from the Command Line

To change the references, perform the following steps at the UNIX command line:

1. Set the working directory to the directory above your design directory. This enables you to globally change references in the design directory.
2. Type **`listref`** ***`design directory`*** to look at the path references to the libraries.
3. Type **`chref xc3000 xc4000`** ***`design directory`*** to change references from XC3000 library to XC4000 library.
4. Type **`listref`** ***`design directory`*** to check the changes to the path references to the libraries.
5. Open your design in Design Architect and perform a check sheet on each schematic sheet. Correct any errors that might have occurred during the retargeting.

Mentor Graphics Interface/ Tutorial Guide

Design Techniques

Printed in U.S.A.

Chapter 3

Design Techniques

This chapter provides general information and techniques you should be familiar with when you create both FPGA and EPLD designs using Design Architect.

Naming Conventions

It is strongly recommended that you label all nets, symbols, and buses to make debugging and simulation easier. The following naming conventions must be used when designing a PLD with Design Architect. If you use an illegal signal name, you may get warning messages when the design is processed. If this occurs, return to the schematic to fix the name. For the naming conventions of programs not discussed in this section, refer to the *XACT Reference Guide*.

- All names given to symbols and signals in a PLD design must be valid for the XACT Development System. User-defined names can only contain: A-Z, a-z, 0-9, _, -, <, and >.
- Names must contain at least one non-numeric character.
- Directory names must not begin with numbers.
- Names cannot be more than 1024 characters long.
- You cannot used XACT-reserved names. For example, for FPGA designs, you cannot use CLB names, such as AA and AB; PIN names such as P1 and P2; and PAD names such as PAD1 and PAD2. For EPLD designs, you cannot use “PRLD”, “MRESET”, or any name beginning with “_N” (underscore N) as a net label.
- Mentor Graphics-reserved names such as “Input,” “Output,” and “Latch” are invalid.

EDIF2XNF

The EDIF2XNF program translates an EDIF file into an XNF file. Although EDIF2XNF accepts different variations of EDIF files, this section is limited to EDIF files created with Mentor Graphics ENWRITE program. EDIF2XNF keeps the full path names for all signal and symbol names. Do not use # or ~ because EDIF2XNF does not check for name contention.

The EDIF2XNF program expands bus notation. All bus and symbol pin names are expanded into individual signal or pin names. For example, a bus labeled DATA(3:0) is converted into four nets labeled DATA<3>, DATA<2>, DATA<1>, and DATA<0>.

The following table lists examples of legal bus names.

Table 3-1 Bus Name Examples

Bus Name	Description
Q(0:7)	8-bit bus, signals Q(0) (MSB) through Q(7) (LSB)
Q(7:0)	8-bit bus, signals Q(7) (MSB) through Q(0) (LSB)

Because Design Architect and XNF netlists have different naming conventions, EDIF2XNF converts characters according to the following table.

Table 3-2 Symbols Converted By EDIF2XNF

Design Architect Name	XNF Name
#	- (hyphen)
~	- (hyphen)
(	<
)	>
[	<
]	>

Signal Naming Conventions

The following are EDIF2XNF signal naming conventions:

- Hierarchical signal names are fully specified in the XNF file. For example, “ABC” represents a signal named “ABC” in the top level of a schematic drawing. Another example is “I$23/ABC,” which represents a signal named “ABC” under instance “I$23.”
- Unnamed signals are given default names in the form **N$***number*, for example, N$12, N$142.

Symbol Naming Conventions

The following are EDIF2XNF symbol naming conventions:

- To name a symbol, add an INST property. Each INST value must be unique in a schematic. Lower-level symbols have hierarchical path names appended to their instance names, which ensures the uniqueness of the symbol names. For example, “mysym” represents a symbol with an INST property of “mysym” located at the top level. The notation “/top/mysym” represents instance “mysym” located one level below “top.”
- Symbols without an INST property are given default names in the form **I$***number*, for example, I$1, I$15.

Xilinx Libraries

In Design Architect, the XACT Libraries menu contains the Unified Libraries and the Obsolete Libraries. The Unified Libraries are a collection of libraries that conform to standards set for the appearance, function, and naming conventions of the library elements. This standardization allows you to easily convert from one Xilinx architecture to another. You should use the primitives and the macros in the Unified Libraries to create new designs.The Obsolete Libraries contain pre-Unified Libraries primitives and macros and are provided for backward compatibility. Creating a new design from the Obsolete Libraries is not recommended since support for this library will be discontinued in the future. Refer to the *XACT Libraries Guide* for detailed information on the Xilinx Libraries.

Primitives and Macros

The XACT Libraries contain three types of components: primitives, soft macros, and Relationally Placed Macros (RPMs). Primitives are pads and basic logic elements, such as gates, latches, flip-flops, buffers, and oscillators. Soft macros are schematics that contain primitives and other soft macros. Soft macros have pre-defined functionality, but have flexible mapping, placement, and routing. RPMs, available for XC4000 devices, are soft macros that contain placement information and that can contain carry-logic elements.

Note: User-generated hard macros from pre-Unified Libraries designs must be converted to RPMs with the HM2RPM program. Refer to the *XACT Reference Guide* for detailed information on hard macro conversion.

X-BLOX

Note: The X-BLOX library cannot be used to create EPLD designs.

The X-BLOX library contains module generators that describe a system using high-level functions instead of gate primitives. The X-BLOX synthesis tool processes these modules. The X-BLOX library can be used with XC3000A, XC3000L, XC3100A, and XC4000 FPGA designs. See the *X-BLOX User Guide* for more detailed information.

Using the XACT Libraries

The following procedure describes selecting a component from the Unified Libraries and placing it in your schematic. Do not mix components from different families, and do not mix components from the Obsolete Libraries with components from the Unified Libraries.

From within Design Architect, select and place library components as follows:

1. Select **XACT_LIB** from the Libraries pull-down menu. The schematic palette is replaced by the XACT libraries menu palette.
2. Select **UNIFIED LIB** from the libraries menu. The XACT Unified libraries menu for different part families appears.
3. Select the correct library for your design. A menu appears and you can select **BY TYPE** or **ALL PARTS**. If you select by type, a list of the components organized into categories such as buffer, counter, or flip_flop appears. If you select all parts, all the components are displayed in alphabetical order. Use the Page Up and Page Down keys to move up and down the list of components.
4. Select a component from the library list. A small dialog box appears on the screen.
5. Move the cursor into the schematic window. An outline of the selected component appears.
6. Move the outline to the appropriate location and click the left mouse button to place the component.

Bus Rippers

Bus rippers are Mentor Graphics-supplied special components that connect nets to specific signals on a bus. You can obtain bus rippers by selecting the rip component in the Logic submenu in the Unified Libraries. These components are the same as rip components in the MGC Digital Libraries gen_lib.

A bus ripper consists of two pins. The narrow end is the wire end and the wide end is the bundle end. The wire end always connects to a net or smaller bus, and the bundle end connects to a bus. The bus ripper can tap all or a set of signals into a new bus. Refer to the following figure for an example of a bus ripper.

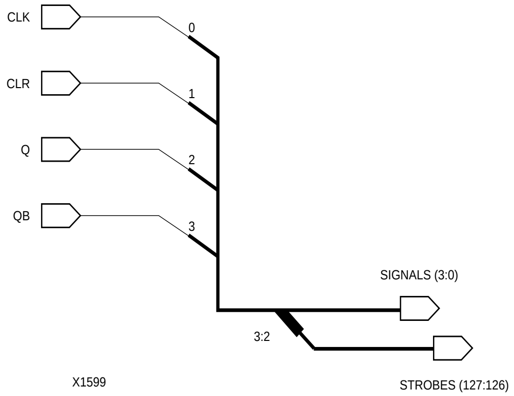


Figure 3-1 Bus Ripper

The ripper has a RULE property, which defines the bit or bits being tapped from the bus. By default, the RULE property is set to R, but you must change the property value to represent the bit or bits you want tapped from the bus.

To change the property value, perform the following procedure:

1. Select the wire end of the bus ripper part whose RULE property you want to change.
2. Access the Edit Window pop-up menu and select **`Properties`** → **`Modify`**.
3. Select the **`RULE`** property and enter the desired property value in the Property Value box. For more information on bus rippers, refer to the tutorials in this manual and the *Design Architect User's Manual.*

Creating a Schematic

The following steps describe a simplified procedure for creating a design in Design Architect. For detailed instructions refer to the Design Architect tutorial in this manual and to Mentor Graphics *Design Architect User's Manual.*

1. Use the Open Sheet icon in the Session Palette to open a new sheet.
2. Place Unified Libraries components and user-created symbols in the new sheet. If you create your own symbols, you must also create the underlying schematics.
3. Add and label nets and buses.
4. Add or modify schematic and symbol attributes as described in the next section.
5. Check and save your design.

Entering Xilinx Attributes

Although a few differences exist when comparing PLD designs to other ASIC or board-level designs, PLD schematic design generally involves the same techniques used when designing other technologies. Most of these differences involve adding Xilinx PLD-specific attributes to schematic components. This information is used by the design implementation software during placement and routing of your design.

In Design Architect, adding Xilinx attributes is called property annotation. Property annotation is used to add design information called “properties” to schematics and symbols. These added properties describe characteristics of a component that are not identifiable from the schematic drawing alone. A description of FPGA-specific attributes is included in the chapter on FPGA design issues. EPLD-specific attributes are described in the chapter on EPLD design issues.

The method used to add or modify component attributes is identical for FPGAs and EPLDs as described below:

Adding Properties

To add a property name and value to a selected object in the schematic editor window, perform the following procedure.

1. Select object(s), for example: net, pin, or instance.
2. Press the Shift-F5 function key. The Add Property dialog box is displayed.

3. Scroll through the list of existing property names and click with the left mouse button on the property name. To add a new property name, enter the new name in the Property Name box.

Note: If you want to add a property that is not a Xilinx attribute, you must include an equal sign before the property name in the property name box. This ensures that user-defined properties are passed to the XNF file. For example, to add the user-defined property "group1=FFS(A*:B*)", enter "=group1=FFS(A*:B*)".

4. Type the new property value in the Property Value box.

Note: For some FPGA properties, the property name and the property value are identical. See the "FPGA Design Issues" chapter for more information.

5. Fill in the rest of the dialog box and then click the **OK** button.The **`ADD PR`** prompt bar appears.
6. Move the cursor to where you want to place the property text and click the left mouse button to place it.

Modifying Properties

To modify the default property value of a selected object in a schematic editor window, perform the following steps:

1. Use the right mouse button to display the popup menu. Select **`Properties → Modify`**. A dialog box appears listing the properties of the selected object.
2. Select the property you want to modify and click the **OK** button.
3. The Modify Property dialog box is displayed. Enter the desired changes and then click on **OK**.

Merging Design Files from Other Sources

You can enter part of your design in some form other than schematics, such as text entry or a RAM or ROM description. You can also bring in netlist files produced by interface software other than Mentor Graphics. Whatever the form of entry, the starting point for inclusion into a Mentor Graphics schematic design must be a netlist file in XNF format. One exception is that equation-based logic for EPLD designs is read directly by the XEPLD Fitter in the form of

VMH / VMD files rather than being merged into the XNF netlist.XNF netlist files must be located in the working directory. Without the XNF file, this portion of the design cannot be included; with it, the origin of the logic becomes irrelevant. To incorporate the XNF file into your schematic, you must create a symbol for the file and place it on your schematic as you would any other component. To incorporate a MemGen file into your schematic, manually create a symbol in Design Architect and then add the FILE property to the symbol to specify the MemGen file.

Creating a Xilinx ABEL Symbol

An XSF file and either an XAS file (FPGA) or a PLD file (EPLD) are generated by Xilinx ABEL from the file containing the ABEL-HDL description of the logic. The XSF file is used to create the symbol that represents the underlying logic description contained in the XAS or PLD file. The XSF file contains the pinouts for the symbol. The Gen_Sym8 program automatically creates a symbol based on information in the XSF file.

Note: The XSF file must be located in your working directory.

To create a symbol from an XSF file, perform the following:

1. At the UNIX system prompt in your project directory, type the following to confirm that the Xilinx ABEL XSF file is in the project directory.

   ls *.xsf

2. Enter:

   gen_sym8 *design*_abl.xsf

   Gen_Sym8 executes, reads the XSF file, and writes out the symbol $MGC_WD / *design*_abl.

Adding a Xilinx ABEL Symbol to Your Schematic

1. Execute PLD_DMGR to enter the Mentor Graphics Design Manager.
2. Select your design in your project directory and open it in Design Architect.

3. Select **`Right Mouse Button → Instance → Symbol by Path.`** A dialog box appears.
4. Use the navigator button in the dialog box to select the symbol created in the above procedure or type the symbol name in the Component Name field.
5. Select **`OK`** or press return.
6. Place the symbol by moving the cursor to the appropriate location and clicking the left mouse button.
7. For FPGA designs, add the FILE property to the symbol as described in the "Adding Properties" section of this chapter. Enter FILE as the property name and *design*_abl as the property value. The value of the FILE property specifies the name of the XAS file containing the logic description. For EPLD designs, see the next step.
8. For EPLD designs, perform the following:
   - Add the FILE property to the symbol as described in the "Adding Properties" section of this chapter. Enter FILE as the property name and <*design*> as the property value. The value of the FILE property specifies the name of the PLD file containing the logic description.

Note: The following two steps are optional for EPLD designs.

- Add the PLD=<*design*> property to the symbol to allow PLD_XEMake to run PLUSASM automatically.
- Add the DEF=pld property to the symbol to tag it as unsuitable for functional simulation.

As an alternative to creating a custom symbol, you can use one of the standard or generic PLD symbols provided in the XC7000 library as a convenient way to represent equation-based logic in your schematic.

Mentor Graphics Interface/ Tutorial Guide

FPGA Design Issues

Chapter 4

FPGA Design Issues

This chapter provides FPGA-specific design information and techniques you should be familiar with when you create your schematics with Design Architect. For more detailed information on Xilinx attributes, refer to the *XACT Libraries Guide*.

FPGA Properties

Add or modify the FPGA properties described in this chapter to specify Xilinx attributes for the elements in your designs. These attributes provide information to the implementation tools during the processing of your schematic design. Instructions for adding or modifying properties are included in the "Design Techniques" chapter.

Symbol Properties

Use the following properties to specify symbol attributes.

REF

Note: In V7, the COMP property was used to name a model; in V8, use the REF property. Do not use COMP, because it indicates that the model is a library primitive. The ENWRITE netlister stops expansion at all COMP models and does not merge in your lower-level schematic.

Use the REF property to name a symbol. You can have multiple symbols with the same REF property.

PINTYPE

Add the PINTYPE property to a pin to identify it as input or output for PLD_DVE. PLD_DVE uses the PINTYPE property to determine the pin directionality of all of the symbol's pins. Any symbol with a FILE property also requires PINTYPE properties. When adding PINTYPE properties, select **`PINTYPE`** from the list of properties and type **`in`**, **`out`**, or **`io`** for input, output, or bidirectional, respectively in the value box.

INST

Use the INST property to uniquely identify a symbol in a design. Design Architect assigns a default INST property to each symbol (`I$1`, `I$2`, and so forth), and the INST value is appended to the hierarchical path.

COMP

Use the COMP property to indicate that a simulation model exists for a primitive. All Xilinx primitives have a COMP property.

Warning: Never add a COMP property to a hierarchical block.

In V7, the COMP property named a model. In V8, use the REF property to name a model; do not use COMP, because it indicates the model is a library primitive. The ENWRITE netlister stops expansion at all COMP models, and does not merge in your lower-level schematic.

FILE

Use the FILE property on a user-created symbol to indicate the name of the XNF file that represents the underlying logic. When adding the FILE property, type the name of the XNF file in the Property Value box; do not include any extensions. The file must exist in the directory where PPR or XNFMerge is run. The file name and symbol name must match exactly, including case. For example, if the file name is "andblk.xnf", the symbol name must be "andblk"; if the file name is "ANDBLK.XNF", the symbol name must be "ANDBLK". Do not use the FILE property on primitives.

If you use the FILE property on symbols that are not in the same directory as the top level design, you must use the Auto Generate option when you run PLD_FNCSIM8 and PLD_TIMSIM8. Refer to the "Functional Simulation Preparation" and "Timing Simulation Preparation" chapters for more information on the Auto Generate option.

The FILE property allows you to perform map-then-merge in non-schematic portions of the design. Map-then-merge maps each functional block to its own CLBs. If you want a functional block mapped separately, or if only an XNF file exists, you must create a symbol for that functional block with the FILE and PINTYPE properties.

BLKNM

Add the BLKNM property to symbols to ensure that the symbols are included in the same CLB, IOB, or BUFT after logic partitioning.

Note: The BLKNM property is not a LOC value. It simply groups symbols with the same BLKNM into the same CLB or IOB.

Use the same BLKNM value for each symbol you want to group. The BLKNM value can be any alphanumeric string composed of A-Z, a-z, 0-9, _, -, <, and >. By default, one of the output signals of the CLB is used for the BLKNM value for that CLB. You can use the BLKNM property to attach a name to any of the following symbols:

- XC4000 components, flip-flops, and I/Os
- XC3000 flip-flop components (FD, FDC, and FDCE symbols)
- XC2000 flip-flop and latch components (FD, FDC, FDCP, LD, LDC, LDP, and LDCP symbols)
- I/O block primitives (IOB symbols)
- XC3000 configurable logic blocks (CLB symbols, CLBMAP symbols)
- XC4000 configurable logic blocks (FMAP and HMAP symbols)
- 3-state buffers (BUFT symbols)

HBLKNM

Note: Do not use HBLKNM to group BUFTs.

This property is similar to BLKNM. The name value of the BLKNM property remains unchanged throughout the design process, however, the name value of the HBLKNM property is hierarchically qualified during the design-flattening process. The name value for HBLKNM has each level of the design hierarchy prefixed to the name, with each level separated by a "/". If two or more symbols have the same hierarchically qualified HBLKNM value, the logic mapping software will attempt to place those symbols in the same CLB or IOB.

MAP

Use the MAP property to control logic partitioning. Attach this property to FMAP, HMAP, or CLBMAP symbols to specify that pins can be swapped during routing and that logic, other than what has been explicitly specified, can be merged into a CLB or function generator.

DOUBLE

On the XC3000 and XC4000 family parts, internal bus structures with programmable pull-up resistors are available for implementing bidirectional buses or wired-AND functions. Two pull-up resistors are available on each internal bus line. You may use both resistors for a fast, high-power signal, or only one for a slow, low-power signal. Add the DOUBLE property to the PULLUP symbol to specify the fast, high-power option. The slow, low-power option is used by default. The DOUBLE property has no effect on PULLUP symbols that are used to specify a pull-up resistor on an external signal.

When you add the DOUBLE property, enter **`Double`** in the name box as well as in the value box.

DECODE

Attach the DECODE property to the body of XC4000 WAND symbols to indicate that the wired-AND is implemented in edge decode logic.

When you add the DECODE property, enter **`Decode`** in the name box as well as in the value box.

INIT

Note: The INIT property is required for the ROM symbol.

Use the INIT property on the ROM symbol. The value will be 4 or 8 HEX characters. The Partition, Place, and Route (PPR) program reads in the INIT property on the ROM and translates it into the appropriate logic gates. Because PPR hard-codes the initial value into the LCA file, you cannot change the initial value after running it. Unlike RAMs, ROMs can have pre-programmed data in the Xilinx part. The RAM primitive cannot be initialized during configuration and must be written to after the device is configured.

EQN

Use the EQN property only on EQN symbols to specify the function of the symbol without having to use the equivalent gate level logic.

DEF

Use this property on all X-BLOX symbols to indicate that the symbol represents special logic.

CYMODE

Use the CYMODE property on the Carry Mode symbol to identify the mode for the dedicated carry logic in an XC4000 CLB.

SCHNM

Use this property to carry forward the original symbol type name from the schematic library into the XNF file.

LIBVER

This property is on all Unified Libraries symbols to distinguish netlists created using the Unified Libraries from those created with earlier versions of the libraries.

Location Properties

Use the following properties to specify location attributes.

LOC

You can use the LOC property to specify a location where a symbol can or cannot be placed. When the LOC property is placed on macros, the property is passed down to the primitive level. If more than one LOC property is placed on different levels of hierarchy, the property placement on the highest hierarchical level is recognized and all lower-levels are ignored.

You can use the LOC property on the following symbols:

- XC4000 flip-flop soft macros (FD symbols)
- XC3000 flip-flop soft macros (FD symbols)
- XC2000 flip-flop and latch components (FD symbols)
- I/O buffers (IOB symbols)
- I/O block primitives (IOB symbols)
- Configurable logic blocks (CLB symbols, CLBMAP symbols, FMAPs and HMAPs)
- 3-state buffers (BUFT symbols)
- XC3000/XC4000 horizontal longline pull-up resistors (PULLUP)

For symbols that map to single CLBs or IOBs, the value must be a valid CLB or IOB name, respectively. In the *XACT Reference Guide,* refer to the APR chapter for information on XC3000 legal location names and the PPR chapter for XC4000 legal location names. The LOC property can also be used for logic that uses multiple CLBs, IOBs, pull-up resistors, BUFTs, soft macros, primitives, or other symbols.

Note: You cannot add the LOC property to a pad, however, you can add this property to a pad's external net by selecting the net vertex and following the procedure below.

Perform the following steps to add the LOC property to symbols in your design:

1. Select the symbol.
2. Display the Edit window pop-up menu and select **`Properties`** → **`Add`**. A dialog box appears.
3. Type **`LOC`** in the Property Name box.
4. Enter the desired value in the Property Value box.
5. Select **`String`** in the Property Type box.
6. Click on **`OK`** and place the text in the desired location.

Note: The following location properties are used to specify, name, and manipulate sets of symbols that have relative location constraints. They can be added to flip-flop primitives, memory, carry logic, BUFTs, non-DECODE WANDs, WORANDs, mapping symbols, and macros.

RLOC

Note: The RLOC property can only be added to primitives.

Use the RLOC property to define the desired row and column relationship between two or more symbols. This property defines the spatial relationship between two or more symbols, not the absolute location. RLOC property values are propagated down the design hierarchy.

USE_RLOC

Use the USE_RLOC property on a macro symbol to tell the merge program to use or ignore the RLOC information inside the macro symbol.

U_SET

Use the U_SET property to specify symbols that belong to a unique set of objects that have a relative location relationship. This parameter can be attached to primitives that have an RLOC property as well as to hierarchical symbols that have primitives with the RLOC property below them in the hierarchy. The name field of the U_SET property does not change during the design flattening process.

HU_SET

The HU_SET property is similar to U_SET except it works within the bounds of the design hierarchy. All of the symbols that belong to each unique HU_SET must be located within the same branch of hierarchy. The name field of HU_SET has each level of the design hierarchy prefixed to the name, with each level separated by a "/".

RLOC_ORIGIN

The RLOC_ORIGIN property fixes the absolute location origin for a set of symbols that have a relative location relationship.

RLOC_RANGE

The RLOC_RANGE property specifies the range of FPGA locations that are allowed for a set of symbols with the RLOC property. Unlike the RLOC_ORIGIN property that fixes each member of a set at an absolute location, the RLOC_RANGE property allows the members of the set to be located anywhere within the range, as long as the RLOC relationships are maintained.

Input/Output Properties

Input/Output properties are only allowed on I/O symbols or on the EXT record that corresponds to the external connection of the I/O symbol.

Note: The TTL and CMOS properties are not allowed on EXT records.

INTERNAL

Use the INTERNAL property to identify unbonded IOBs.

NODELAY

Add the NODELAY property to XC4000 I/O flip-flop or latch elements, such as IFD and ILD, to remove the delay element and reduce setup time.

FAST or SLOW

Use the FAST property on XC3000 and XC4000 output symbols or pads to indicate that the output driver of the corresponding IOB will not be slew-rate limited. The FAST property can be attached to the net of any OPAD or the body of an OBUF, OBUFT, OFDT, or OFD to increase the speed of the IOB. The default value is SLOW. The FAST property decreases the output drive transition time, but increases noise. When you add the FAST property, enter **`Fast`** in the name box as well as in the value box.

MEDFAST or MEDSLOW

Use the MEDFAST or MEDSLOW property on XC4000A output symbols to specify the slew rate of the output driver. The MEDFAST property decreases output drive transition time and increases noise, but not as much as the FAST property. The MEDSLOW property decreases output drive transition time and increases noise, but not as much as the MEDFAST property. When adding these properties, enter **`Medfast`** or **`Medslow`** in the name box as well as in the value box.

RES or CAP

Add the RES or CAP property to the output driver of an XC4000H IOB to specify resistive mode or softedge capacitive mode, respectively.

CMOS or TTL

Add the CMOS or TTL property to XC4000H IOB symbols to define the input sensing and output driving levels as either CMOS-compatible or TTL-compatible, respectively.

CLB/IOB Properties

BASE

Use this property to specify the block base value of CLB and IOB symbols. Enter a value, depending on the family, in the Value box when you set the CLB or IOB primitive properties.

CONFIG

Use this property to specify the connectivity within the CLB or IOB. The CONFIG property value is composed of multiple tags that represent each programmable option.

EQUATE_F AND EQUATE_G

Note: The EQUATE_F and EQUATE_G properties can only be used with CLBs.

The EQUATE_F and EQUATE_G properties specify a logic equation that describes the F or G outputs of a CLB. An EQUATE value consists of a Boolean expression that is composed of CLB pin names and Boolean symbols.

The EQUATE_F and EQUATE_G properties set the equations for the function generators and must have an equation as their value. You can use parentheses in the equation. Use CLB pin names, not net names, in the EQUATE statement.

Modifying CLB/IOB Properties

To modify the properties of a CLB or IOB symbol, perform the following procedure:

1. Select the component.
2. Bring up the pop-up menu and select **Properties → Modify**. A dialog box appears.
3. In the dialog box, select the properties you want to modify. You can select more than one property by holding down the Control key while selecting the property names. After specifying all properties, click on **OK**. A dialog box appears for the first property you selected.
4. Modify the property, then click on **OK**. A dialog box appears for the second property you selected. Repeat this process until you modify all the selected properties.

Timing Specification Properties

Note: You can only specify timing constraints on XC3000A/L, XC3100A, and XC4000 designs.

Specifying timing requirements at the schematic level helps ensure that a design functions properly after it has been implemented into a device. The Partition, Place, and Route (PPR) program uses timing-constraint information when it places and routes a design. PPR uses the timing specifications from the schematic to calculate which nets should have the least amount of slack during the place and route process. PPR uses the slack calculations to place logic functions and to measure its progress against the timing specifications you supply. For more information on timing specification, refer to the XACT-Performance section in the *XACT Reference Guide*.

TS*identifier*

Add the TS*identifier* property to TIMESPEC symbols to convey timing information to the place and route program.The *identifier* is a name consisting of any combination of letters, digits, or underscores.

TNM

Note: The TNM property is not allowed on I/O pads from the Unified Libraries, however, you can attach it to nets connected to I/O pads.

Use the TNM property to define groups of path endpoints for use in TIMESPEC specifications. The TNM property is allowed on I/O flip-flops, CLB flip-flops, I/O latches, CLB RAMs/ROMs, and EXT records, as well as on macro symbols. When a TNM is specified on a macro symbol, it will be propagated down to all lower-level symbols of the appropriate type by the merge program.

TS Flag

Use the TS flag to attach timing information from a non-default TS property to a net in the schematic.

Adding TIMESPEC/TIMEGRP Properties

Note: In the following steps TIMEGRP can be substituted for TIMESPEC.

Perform the following steps to add TIMESPEC properties:

1. Select the TIMESPEC primitive.
2. Select **Properties → Add → Add Multiple Properties** from the sub-menu. A dialog box appears.
3. In the Property Name field, enter a **TS***identifier. Identifier* is an alphanumeric field. For TIMEGRP symbols, enter a leading equal sign before the property name.

Note: TS*identifier* is not visible in the TIMESPEC primitive when using Design Architect. To view a list of identifiers and their corresponding values, select the TIMESPEC primitive, press the right mouse button, and select **Properties → Modify** from the pop-up menu.

4. Enter the TS attribute definition (for example, DP2P:100) in the Property Value field.
5. Repeat steps 3. and 4. until you define all TS attributes. Use another TIMESPEC primitive if you have more than eight TS attributes in your design.
6. Select **String** in the Property Type field, select **On** in the Visibility field, and select **Add Property to Vertices** in the remaining field.
7. Click on **OK**.
8. Place the TS attributes in the fields within the TIMESPEC primitive.

Adding a TS Flag to a Net

Perform the following steps to attach a TS flag to a net in your schematic:

1. Select a vertex of the net to which you want to add the TS flag.
2. Bring up the pop-up menu and select **Properties → Add Single Property**. A dialog box appears.
3. Enter **Netflag** in the Property Name field.
4. Enter **TS***identifier* in the Property Value field, where *identifier* is an alphanumeric field.
5. Select **String** in the Property Type field.

6. Select **`On`** in the Visibility field.
7. Select **`Attach to Vertices`** in the remaining field.
8. Click on **`OK`**.
9. Place the TS flag on the schematic.

Net Properties

NET

Use the NET property to connect hierarchical levels or multiple sheets, in addition to using them during simulation. To add a NET property, follow the steps provided in the *Design Techniques* chapter or use the following method:

1. Select all of the net vertices that you want to name.
2. Select **`Name Nets`** from the pop-up menu, name the first net in the dialog box that appears, then place the property text.
3. Subsequent dialog boxes appear for each net you selected; name and place the property text for each one.

NETFLAG

Use the NETFLAG property on nets to affect the partitioning, placement, and / or routing of a design. You can attach only one NETFLAG property to one vertex per net. If you add the Netflag property to more than one vertex on a net, XNFMAP or PPR arbitrarily recognizes only one of them.

In XC4000 designs, you can use the NETFLAG property to identify the timing requirements of the attached net. Every net carries a routing priority or net weight from 0 to 100. If you do not specify a net weight by using the NETFLAG property, the default weight of 1 (one) is assumed.

A NETFLAG property can have any of the following values:

- X - External. The External flag ensures that the selected net is not partitioned within a CLB. The partitioning routine within XNFMAP or PPR absorbs nets as it reduces the combinatorial logic of a design.

- C - Critical. The Critical flag defines a net as critical (net weight = 10 for APR, 100 for PPR). The Automated Design Implementation software (ADI) tries to minimize delays on this path by routing this net first. In an XC4000 design, the critical flag only applies if the PPR constraint PATH_TIMING is set to false; false is not the default.
- N - Non-critical. This flags a net as non-critical (net weight = 0). ADI gives this signal lowest routing priority. In an XC4000 design, the non-critical flag only applies if the PPR constraint PATH_TIMING is set to false; false is not the default.
- L - Longline. The Longline flag instructs ADI to use a Longline to route this net. This Netflag value is useful for nets with high fan-out that need low skew (XC2000 and XC3000 family only).
- I - This value connects any CLB clocks driven by this net to the C input (XC2000 family only).
- G - This value connects any CLB clocks driven by this net to the G function output (XC2000 family only).
- K - This value connects any CLB clocks driven by this net to the K input (XC2000 family only).
- S - Save. The Save flag prevents XNFMAP from removing unconnected signals. If you do not have the S value attached to a net, XNFMAP removes any signal not connected to logic and/or an I/O primitive and keeps it external to a CLB.
- P - Pinlock. The Pinlock flag (CLBMAP primitives only) ensures that the Automated Placement and Routing (APR) software does not move the CLBMAP pin to which the property and signal are attached; this is useful for aligning CLB inputs with a specific Longline.
- W=# - Weight. This value assigns a weight to a net, indicating its routing order. Legal weight values are 1-99, where 99 indicates the most critical net, the highest in the routing order. This attribute affects the partitioning algorithm and is also used by the place-and-route routines (XC4000 only). For XC4000 designs, the weight flag only applies if the PPR constraint PATH_TIMING is set to false; false is not the default.
- SC - Skew Critical. This flag indicates that the signal is skew critical and that the software must minimize the difference

between load delays on this net. This value can be used with the net weight (W=) value (XC4000 only). In an XC4000 design, the skew-critical flag only applies if the PPR constraint PATH_TIMING is set to false; false is not the default.

When adding the NETFLAG property, enter the property value: X, C, N, L, I, G, K, S, W=*weight*, or SC in the Value box. Valid combinations of values must be separated by commas. For the Pinlock property, select the CLBMAP pin and select **`Properties`**→**`Modify`** from the Edit Window pop-up menu. Set the value to **`y`** (XC2000, XC3000 only).

Mentor Graphics Interface/ Tutorial Guide

EPLD Design Issues

Chapter 5

EPLD Design Issues

This chapter explains how to create your schematic design so it can be fitted to an EPLD device. It covers the following topics:

- Using the schematic library components, and how to take advantage of EPLD features.
- Using the schematic attributes to control logic optimization and other fitting options.

Components

All the components that can be used in EPLD designs are contained in the XC7000 library. Most of the components can be used for either EPLD or FPGA designs. A few components are specific to EPLD. This chapter describes how to use the common and EPLD-specific components.

There are three basic types of components:

- Buffer or Pad — These components define input and output ports that represent physical pins on the device.
- Standard — These components represent fixed logic functions such as gates, adders, counters, and so on.
- PLD — These components are user-defined via a PLUSASM equation file.

Many of the components in the XC7000 library are implemented using special features that take advantage of EPLD architecture. The following sections describe some of those features.

Buffers and Pads

Input and Output Buffer Connections

To represent an ordinary device input pin, use an IPAD connected to one IBUF buffer; the IBUF can then connect to any number of on-chip logic symbol inputs as shown in the figure below. IBUF can drive clocks and 3-state output enables (except OBUFEX1), but there are also special-purpose buffer symbols in the library (BUFG and BUFFOE) that you can use instead for these functions.

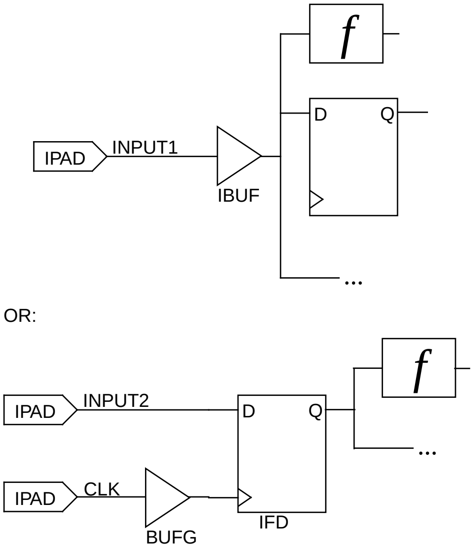


Figure 5-1 Input Buffers

To take advantage of the input-pad registers and latches available in EPLD devices, use one of the IFD, IFDX1, or ILD symbols instead of the IBUF (do not connect an IBUF to the D-input of an IFD/ILD symbol). Refer to the *XACT Libraries Guide* for specific application rules for the symbols.

To represent an ordinary device output pin, use an OBUF buffer that is driven by one (and only one) on-chip logic source. Connect the output of the OBUF to an OPAD symbol. You could also use one of the 3-state output buffers (OBUFE, OBUFT) instead of OBUF. Drive the output enable control input (E or T) using any on-chip logic source or input signal (from IBUF). The EPLD fitter will look for opportunities to automatically assign the enable signal to one of the EPLD's FOE global enable lines.

If you want to take advantage of a FOE global line explicitly, use a BUFFOE input buffer instead of IBUF, and connect it to an OBUFEX1 output buffer (instead of OBUFE) as shown in the following figure.

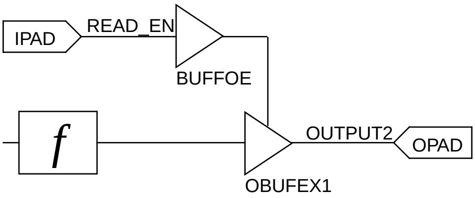


Figure 5-2 Assigning an FOE Line

Note: You should always label all the wires connecting between PAD symbols and input/output buffer symbols. These will be the names by which the software refers to your device pins in the reports and during simulation.

To represent a bidirectional I/O pin, use an IOPAD symbol connected to both the input of an IBUF (or IFD/ILD) and the output of an OBUFE, OBUFT, OBUFEX1, and so on as shown in the following figure.

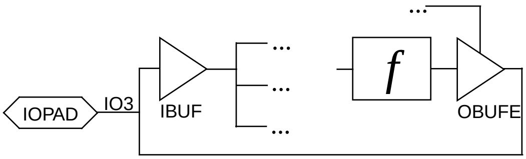


Figure 5-3 Bidirectional Pin

Output Buffers and 3-State Buffers

If a signal going into a common output buffer (OBUF) is generated by any component containing a 3-state buffer (like BUFE or a PLD), the 3-state control signal is used to enable and disable the device output pin driver. This behavior is unique to EPLDs and is not reproduced in FPGAs.

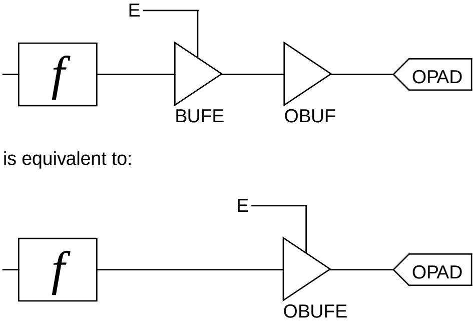


Figure 5-4 Output Enable Behavior in EPLDs

Note: Inserting a buffer (BUF) or inverter (INV) between the 3-state buffer (BUFE) and the output buffer (OBUF) does not prevent the 3-state buffer from controlling the device pin. The XEPLD fitter optimizes all simple buffers and inverters unless you place an OPT=OFF attribute on the 3-state buffer.

If you use a PLD symbol in your schematic and connect one of its outputs to an output buffer like OBUF, you can control the EPLD device output pin using a 3-state control equation in the PLD, as shown in the following figure.

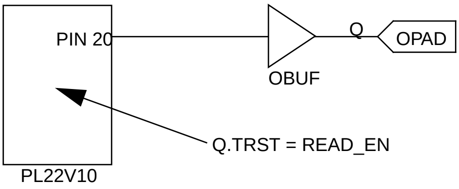


Figure 5-5 Controlling Output Using a PLD Equation

If you want to use a PLD output with a TRST equation to control a bidirectional I/O pin of the EPLD, connect the OBUF output to an IOPAD and IBUF (or IFD/ILD). If the same PLD symbol that generates the output is also to receive the I/O pin input, you must use a separate pin of the PLD symbol to receive the signal from the IBUF. Do not tie the signal received from an IBUF to the wire driving the OBUF of the same IOPAD as shown in the following figure.

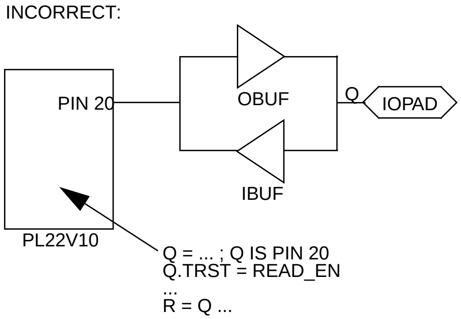


Figure 5-6 Incorrect Way to Control a Bidirectional Pin

These input and output wires must remain separate as shown in the figure below.

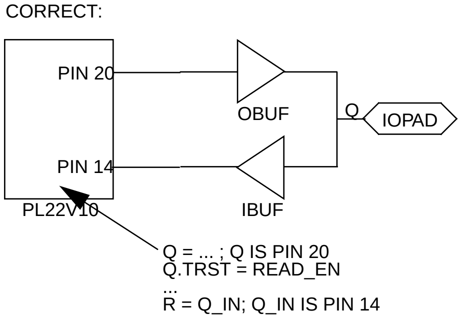


Figure 5-7 Correct Way to Control a Bidirectional Pin

Rules for connecting PLD symbols also apply to any custom symbols defined by equation files or macro schematics.

If your design calls for 3-state multiplexing of multiple output sources, it is best to output each signal source on its own set of 3-state output pins and tie the signals together off-chip. You cannot connect more than one signal source to the same OBUF or OPAD as shown in the following two figures.

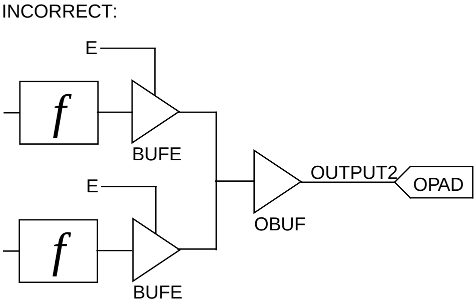


Figure 5-8 Incorrect 3-State Multiplexing

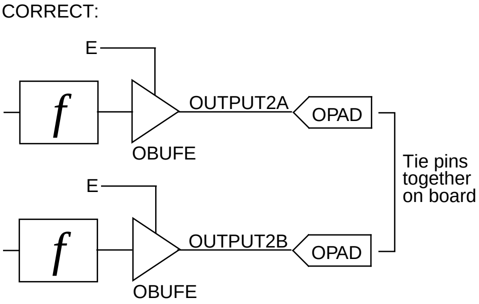


Figure 5-9 Correct Off-Chip 3-State Multiplexing

On-Chip 3-State Multiplexing

EPLD components emulate 3-state signals internally by gating the macrocell feedback to the UIM. (Macrocell feedback signals are never physically in a high-impedance state.) You can tie together the outputs of multiple 3-state buffer symbols (like BUFE or BUFT) or 3-state PLD outputs to multiplex these signals on-chip as in the figure below. Remember that you may not connect such tied signals to an output buffer; you would need to pass a tied signal through a logic symbol (like BUF) before driving an output port.

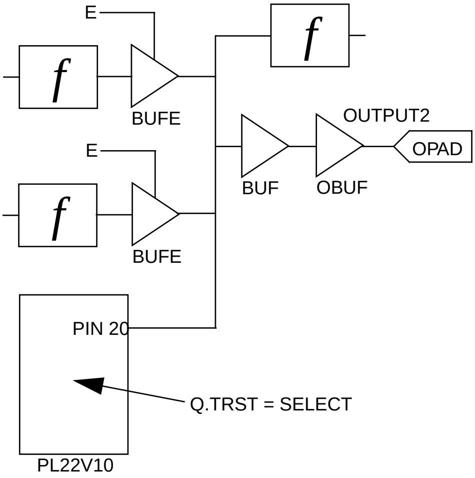


Figure 5-10 Correct On-Chip 3-State Multiplexing

Input Buffers, Clocks, and Global Control Nets

You can connect the clock pin of any FD component or registered component to an ordinary logic signal, an IBUF, or a BUFG (FastCLK) unless otherwise specified in the *XACT Libraries Guide*.

The input of a BUFG component must connect directly to a pad representing a clock pin; there can be no other components between the pin and the BUFG.

IFD and ILD components must have a BUFG clock input.

After assigning any BUFGs to FastCLK pins, the XEPLD software tries to assign IBUF signals driving clock inputs onto FastCLK pins.

The XEPLD software also attempts to optimize FD components into IFDs on the input pads. No other registers are ever optimized into the input pad.

If your design requires a global clock enable, you must use IFDX1 components. The CE input to these components can only be driven by a BUFCE, and the clock must be from a BUFG as shown below.

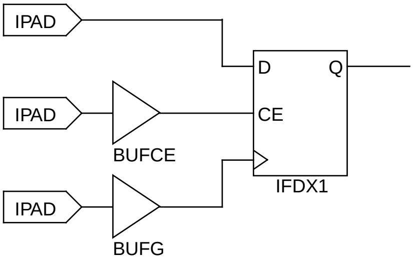


Figure 5-11 Use of the IFDX1 Symbol

Note: You can prevent input register optimization using the REG_OPT=OFF attribute. You can prevent clock optimization using the CLOCK_OPT=OFF attribute.

EPLD-Specific Components

In general, it is best to use EPLD-specific components whenever their features appeal to your application because they are designed to take advantage of special architectural features. For example, EPLD-specific adders take advantage of fast carry chains.

If, however, you want your design to be retargetable to either an EPLD or FPGA device, you should use only the common components. Most EPLD-specific components have at least one common counterpart. The following table provides a summary of Common and EPLD-Specific Symbols.

Note: Each common component functions identically in EPLD and FPGA devices. However, there may be a significant difference in efficiency or performance between the families. Whenever you map a design to a new device, you should do the following:

- Check the reports created during integration carefully to make sure that the way you expressed your design does not consume excessive logic resources of the target device.
- Perform timing analysis or simulation to catch any inefficient parts of the design.

If you need further information on XEPLD software and design technique, see the *XEPLD Design Guide*.

Table 5-1 Common and EPLD-Specific Symbols

Common Symbols	EPLD-Specific Symbols
Accumulators	
ACC1	ACC1X1 or ACC1X2
ACC4	ACC4X1 or ACC4X2
ACC8	ACC8X1 or ACC8X2
ACC16	ACC16X1 or ACC16X2
Adders	
ADD1	ADD1X1 or ADD1X2
ADD4	ADD4X1 or ADD4X2
ADD8	ADD8X1 or ADD8X2
ADD16	ADD16X1 or ADD16X2
ADSU1	ADSU1X1 or ADSU1X2
ADSU4	ADSU4X1 or ADSU4X2
ADSU8	ADSU8X1 or ADSU8X2
ADSU16	ADSU16X1 or ADSU16X2
Counters	
CB2CLED	CB2X1
CB4CLED4	CB4X1
CB8CLED4	CB8X1
CB16CLED4	CB16X1
Input Registers	
IFD	IFDX1
IFD4	IFD4X1
IFD8	IFD8X1
IFD16	IFD16X1
Output Buffers	
OBUFE	OBUFEX1

Counters

Up / down counters in the common library, such as CB8CLED, have a single CEO output, which changes in response to the up / down direction input. Gating of this CEO function in this manner does not allow it to be optimized for zero delay. This means that if common up / down counters are cascaded, they cannot operate at their maximum original frequency.

The EPLD-specific up / down counters, such as CB8X2 have separate outputs for the up and down directions, CEOU and CEOD, that are optimizable. You can cascade these components without impacting their maximum frequency.

The EPLD-specific counter symbols are CB2X1, CB2X2, CB4X1, CB4X2, CB8X1, CB8X2, CB16X1, and CB16X2.

Arithmetic Components

The ADD, ADSU, and ACC type common library components use the EPLD macrocell carry chain between outputs within the same component, but not for cascading. If you cascade these components, the carry signals (CI and CO) go through the UIM and incur a delay. The CI and CO pins may connect to any ordinary logic components or I/O ports, but not to the CI and CO pins of EPLD-specific arithmetic components.

The EPLD-specific adders (with names ending in X1 or X2) use the EPLD macrocell carry chain for cascading without incurring a UIM delay. Their CI and CO pins can only be connected to the CI and CO pins of another EPLD-specific arithmetic component (or the PLFB9 symbol).

The EPLD-specific arithmetic symbols are as follows:

Adders: ADD1X1, ADD1X2, ADD4X1, ADD4X2, ADD8X1, ADD8X2, ADD16X1, ADD16X2, ADSU1X1 ADSU1X2, ADSU4X1, ADSU4X2, ADSU8X1, ADSU8X2, ADSU16X1, ADSU16X2

Accumulators: ACC1X1, ACC1X2, ACC4X1, ACC4X2, ACC8X1, ACC8X2, ACC16X1, ACC16X2

Note: The accumulator symbols are not supported by the XC7272 devices.

PLD Components

The XC7000 library contains symbols representing industry standard Programmable Logic Devices (PLDs) such as the PL22V10 and the PL20V8. When using these PLD devices you must link the PLD component instance in your schematic to the associated PLD equation file (or the imported JEDEC file). The PLD equation file, processed by the PLUSASM assembler, defines the functionality of the PLD component. For instruction on how to link the PLD symbol and its equation file, see the description of the PLD attribute later in this chapter.

Because the functionality of a PLD is defined outside the schematic, designs containing any PLDs cannot be functionally simulated prior to fitting to a device. After fitting, you can perform timing simulation on the completed design.

The PLD components are PL20V8, PL22V10, PL20PIN PL24PIN, PL48PIN, PLFB9, and PLFFB9.

Components Not Supported by Specific Devices

The accumulator components, ACC1X1, ACC1X2, ACC4X1, ACC4X2, ACC8X1, ACC8X2, ACC16X1, and ACC16X2, are not supported by the XC7272 devices.

The following components require features only present in High-Density Function Blocks and are therefore not supported by the XC7318 or XC7336 devices, which contain only Fast Function Blocks:

- PLFB9
- ADD symbols
- ADSU symbols
- ACC symbols
- BUFCE
- BUFT
- IFD
- IFDX1
- ILD

- COMPM
- LD
- FDCP, FDCPE
- OBUFT
- OFDT
- XOR7, XOR8, XOR9

Note: The BUFE components are allowed for external outputs only and must allow FOE optimization.

Primitives and Macros

Each symbol in the library is either a primitive or a macro. The primitive logic components in the XC7000 library are all implemented using PLUSASM equation files included in the software. The single I/O pads, buffers, and input registers are also primitives, but have no PLUSASM descriptors.

The PLUSASM equations defining most XC7000 library components are supplied in the $XACT/examples/behavior/library directory for your reference. You may copy and edit these equation files as a convenient way to implement customized logic components.

A macro is any symbol defined by a Mentor Graphics schematic included in the XC7000 library. The macro schematic contains XC7000 primitives and/or other macro symbols. When the source schematic is read by the software, the macro symbols are expanded into their underlying schematics. The components actually processed and reported by the software are the underlying primitives (referenced by their hierarchical instance names) after macro expansion.

The same symbol may be implemented as a primitive in the EPLD library and a macro in the library for another device family and vice versa.

The only time that it’s important to know about macros is when you are interpreting reports, which always break a macro down into the primitives it contains.

User-Defined Primitives and Macros

You can create custom macro and primitive symbols, then draw macro schematics consisting of XC7000 and/or custom library symbols.

Creating a User-defined Primitive

User-defined primitive symbols are defined very much like PLD library symbols, except that you create your own custom symbol.

To create a user-defined primitive, follow these steps:

1. Create a behavioral description of the primitive in PLUSASM. The CHIP statement should specify the symbol name and the keyword COMPONENT as the target PLD type.
2. Run PLUSASM on the behavioral description to generate a VMH/VMD file in the clib directory.
3. Run Gen_Sym8 on the XSF file to create the primitive symbol.
4. Add the FILE=<*symbolname*> property to the primitive symbol to label the primitive as a user-defined component.

Note: The following two steps are optional.

5. Add the PLD=<*symbolname*> property to the primitive symbol to allow PLD_XEMake to run PLUSASM automatically.
6. Add the DEF=PLD property to the primitive symbol to tag the symbol as unsuitable for functional simulation.

Creating a User-defined Macro

To create a user-defined macro, follow these steps:

1. Create a lower level schematic in Design Architect.
2. Create a user-defined macro symbol in Design Architect.
3. Add the REF=<*design*> property to the symbol to identify the underlying schematic.

Assigning Logical High and Low Values

Unused inputs or symbols should not be left unconnected; warnings will be issued by the software. Unused inputs should be tied to a constant high or low logic level in the schematic.

Specify a constant High logic level by using the VCC symbol from the XC7000 library. Specify a constant Low logic level by using the library GND symbol. As an alternative, you can specify a constant High or Low value by connecting a wire segment to a component pin and labeling the wire VCC or GND.

One exception is that unused pins of PLD symbols (those marked NC in the equation file pinlist) should remain unconnected. Another exception is that unused channels of multi-bit I/O ports and buffers may be left unconnected, for example if only six inputs of an IPAD8 and IBUF8 are used in the design. Inputs tied high or low, or left unconnected, will result in the software removing the logic associated with those inputs.

Attributes

Attributes, which you place in your schematic, allow you to control the following aspects of how the software processes your design:

- Linking of PLD symbols and PLUSASM equation files.
- Pin assignment.
- Power consumption.
- Optimization of logic, registers, and control signals.
- Allocation of Fast Function Block resources.

When you integrate your schematic design using the PLD_XEMake tool, the schematic is converted to a PLUSASM equation file. The PLUSASM statements these attributes produce end up in that equation file.

Attributes are used to express information specific to each design, as opposed to run-time options entered through the PLD_XEMake dialog box. There are two ways that attributes are placed in the schematic:

- Component attributes, such as PLD, OPT, and LOC, affect only the component instances on which they are placed.
- Global attributes, such as PRELOAD_OPT and LOGIC_OPT, affect the entire design and can be placed on any primitive symbol in the schematic; typically a TBLOCK.

Using Attributes

Use the **`Properties → Add`** command to add attributes to symbols and nets. Follow these steps:

1. If you are applying an attribute to a symbol, select the symbol. Be sure nothing else is selected.

   If you are applying an attribute to a net, select the joint where the output of the symbol connects to the net. Be sure you have selected only that joint — a single star should appear at that location.
2. Select the **`Properties → Add`** command from the pop-up menu.
3. A dialog box appears. Type the name of the attribute, for example **`OPT`**, in the Property Name box, and the value, for example **`OFF`**, in the Property Value box. For Property Type, select **`String`**. Select **`OK`** or press **`Return`**.
4. Position the cursor where you want to place the attribute, usually above the component or net.
5. Click the left button to place the attribute.

Global Attributes

The global attributes specific to EPLD designs are as follows:

- LOWPWR=ALL
- LOGIC_OPT=OFF
- MRINPUT=ON
- MINIMIZE=OFF
- UIM_OPT=OFF
- FOE_OPT=OFF

- CLOCK_OPT=OFF
- REG_OPT=OFF
- PRELOAD_OPT=OFF

To avoid confusion with component attributes, you should apply global attributes to a TBLOCK component (a box containing no logic in which you can include comments about the design).

PLD Equation File Name: The PLD Attribute

The PLD=*file_name* attribute on a PLD symbol specifies the name of the file with the logic equations for that PLD. This attribute is valid on custom primitive symbols (target COMPONENT in PLUSASM) and the following PLDs: PL20V8, PL22V10, PL20PIN, PL24PIN, PL48PIN, PLFB9, and PLFFB9.

Do not specify the file extension in the PLD=*file_name* attribute. This file must be in PLUSASM (.pld) or PALASM (.pds) format, although you can start with an ABEL or CUPL file and convert it to PLUSASM or PALASM.

You must also specify this *file_name* as the first parameter of the CHIP statement inside the equation file, as described in the PLUSASM Language Reference section of the *XEPLD Reference Manual*. For example:

```
CHIP file_name PL22V10
```

Within the .pld file, the pin list must contain the names of all the signals connected to all the PLD's pins, in the proper order. For example, if you have the signals shown in the following figure, you must include the following pin list in the equation file:

```
   TITLE CNTR6

   CHIP   CNTR6   P16V8;

;PINLIST (Highest pin number = 20)
x4clk start NC   rd cs NC NC NC NC NC
NC    NC    read c5 c4 c3 c2 c1 c0 NC:
```

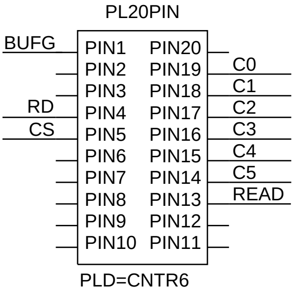


Figure 5-12 Pin List Example

All PLD components in your schematic design must have the PLD attribute. Running PLD_XEMake automatically assembles all equation files named by all PLD=*file_name* attributes found in the schematic.

Like PLDs, user-specified (custom) primitives are defined by PLUSASM equation files. The PLD=*file_name* attribute is not required but can be applied as a convenient way to have your equation file automatically assembled when PLD_XEMake is invoked. If you omit the PLD attribute, PLD_XEMake will expect to find a bitmap file for the symbol (*symbol_name*.VMH or *symbol_name*.VMD) in your local CLIB subdirectory.

Pin Assignment: The LOC Attribute

Use the LOC=*pin_name* attribute on a net connected to a PAD symbol to assign the signal to a specific pin. The PAD symbols are IPAD, OPAD, IOPAD, and UPAD. The pin name is P*nn* for PC packages; the *nn* is a pin number. The pin name is *rc* (rowcolumn) for PG packages. Examples are LOC=P24 and LOC=G2.

Pin assignments are unconditional in that the software will not attempt to relocate a pin if it cannot achieve the specified assignment. You can apply the LOC attribute to as many PADs in your design as you like. However, each pin assignment further constrains the software as it automatically allocates logic and I/O resources to internal nodes and I/O pins with no LOC attributes.

Note: Pin assignment using the LOC attribute is not supported for pad symbols such as OPAD8.

Power Setting: The LOWPWR Attribute

This attribute is valid only for XC7300 designs. You can use this attribute as either a global or component attribute.

The default is LOWPWR=OFF (high speed) for all macrocells used in the design unless otherwise specified.

To make low power the global default power setting, place the global attribute LOWPWR=ALL in the schematic. (See the Global Attributes section above for instructions.)

To determine the power setting of the macrocell(s) used by an individual symbol, use LOWPWR=ON or LOWPWR=OFF (if the global LOWPWR=ALL was used). This attribute is ignored if assigned to a symbol that uses no macrocells, such as an inverter, AND gate (when optimized), input register, and so on.

Logic Optimization Attributes

Use the logic optimization attributes to control optimization of part or all of your design.

OPT

The OPT=OFF component attribute inhibits logic optimization of all macrocells used by a symbol. OPT=ON can override the LOGIC_OPT=OFF global attribute for individual symbols.

The logic optimizer collapses the levels of logic to remove intermediate nodes. Components are optimized forward into components connected to their outputs.

If you build combinational logic using low-level gates and multiplexers, the software attempts to pack all logic bounded between device I/O pins and registers into a single macrocell.

The logic optimizer first removes all internal logic that is not used by any other logic or output buffer and is not explicitly in a PARTITION statement.

The logic optimizer moves logic forward by collapsing combinational expressions into their fanouts. If collapsing an expression into all fanouts succeeds, the original macrocell logic becomes unused and is removed.

The logic optimizer does not collapse an expression into its fanouts if the resulting expression uses too many product terms or inputs.

The logic optimizer also moves forward any logic, whether combinational or sequential, that is buffered by a 3-state buffer. However, logic that itself contains a 3-state control is not moved forward.

The OPT attribute has no effect on any symbol that contains no macrocell logic, such as an I/O buffer.

LOGIC_OPT

To have logic optimization inhibited by default for the entire design, apply the global attribute LOGIC_OPT=OFF. If you do not use this attribute, the default is LOGIC_OPT=ON. You can override the global LOGIC_OPT=OFF for individual symbols using the OPT=ON attribute.

MINIMIZE

Use the MINIMIZE=OFF global attribute to inhibit logic minimization for the whole design. If this attribute is not specified, any redundant or non-effective logic found in any user-specified equation files will be eliminated through Boolean minimization.

UIM_OPT

To inhibit UIM optimization for the entire design, apply the UIM_OPT=OFF global attribute.

UIM optimization extracts AND expressions and inverters out of macrocell logic functions and moves them into the UIM, which reduces the use of Function Block resources.

FOE_OPT

To inhibit FOE (Fast Output Enable) optimization for the entire design, apply the FOE_OPT=OFF global attribute.

FOE optimization generally applies only to BUFE, OBUFE, or 3-state PLD outputs driving an OBUF. FOE optimization changes a product-term 3-state signal to an FOE global control signal. Like FastCLK assignment, this reduces the number of UIM inputs and product terms required by each Function Block.

CLOCK_OPT

To inhibit FastCLK optimization for the entire design, apply the CLOCK_OPT=OFF global attribute.

FastCLK optimization changes a product-term clock to a FastCLK global signal, which reduces the number of UIM inputs and product terms required by each Function Block.

REG_OPT

To inhibit input register optimization for the entire design, apply the REG_OPT=OFF global attribute.

Input register optimization reduces the number of macrocells in a design by moving simple FD registers connected to IBUFs into a pad register (provided that the IBUF has no other fanouts). The clock by which the input register is controlled must be a FastCLK or an input that can be assigned to a FastCLK pin.

PRELOAD_OPT

Apply the PRELOAD_OPT=OFF global attribute to prevent the XEPLD software from changing the preload values of registered components in the design to match the preload values supported by specified device resources such as FFBs and input registers. The default (PRELOAD_OPT=ON) allows the XEPLD software to map your design most efficiently, using the device resources most suited to the elements of your design. Unless you specify PRELOAD_OPT=OFF, the software is free to change the initial register states of any component, including PLD (custom) components defined in PLUSASM. Use PRELOAD_OPT=OFF to preserve the initial states specified in the *XACT Libraries Guide* for library components and in the PRLD equations in your PLUSASM file for PLD or custom components.

You can set a high or low preload for High-Density Function Blocks. The preload value of Fast Function Blocks depends on the use of set or reset. Input register preload values are fixed at 1, except for those on the XC7272, which are undefined.

Fast or High-Density Function Blocks: F and H Attributes

Use NETFLAG=F or NETFLAG=H in an XC7300 device to specify whether a macrocell implementing a component output should be placed in a Fast Function Block (F), or a High-Density Function Block (H). Attach these properties to the joints between output pins and nets.

You can also use the F attribute on the output pin of an IBUF symbol to indicate a Fast Input signal. Only components implemented in Fast Function Blocks receive this signal via the Fast Input path. Any other High-density Function Block components receive the same input via the UIM. Except for using the F attribute on an IBUF symbol, it is not valid to use either F or H attributes on signals originating from any I/O buffer symbol (such as IFD or OBUF).

The F attribute is not valid on outputs of components that require features only present in High-Density Function Blocks, such as the following types of symbols:

- PLFB9
- ADD
- ADSU
- ACC
- BUFT
- COMPM
- LD
- FDCP, FDCPE
- XOR7, XOR8, XOR9

Note: The BUFE symbol can be assigned to an FFB output only when driving an OBUF and must allow FOE optimization.

The H attribute is not valid on outputs of a PLFFB9.

For logic not labeled with F or H attributes, the XEPLD software attempts to put as much logic as possible in the Fast Function Blocks first, then starts filling the High-Density Function Blocks.

MRINPUT

Specifying the MRINPUT=ON global attribute in an XC7354 or XC7336 design changes the Master Reset pin to an ordinary input pin. If this attribute is specified, the EPLD device is initialized only on power-up.

Mentor Graphics Interface/ Tutorial Guide

Functional Simulation Preparation

Chapter 6

Functional Simulation Preparation

This chapter explains how to use the PLD_FNCSIM8 tool to prepare your designs for functional simulation. Functional simulation provides an effective means for identifying logic errors in your design before it is implemented in a Xilinx device. Since timing information for the design is not available, the simulator tests the logic in the design using unit delays. Finding errors before routing your design saves debugging time later in the design process.

The PLD_FNCSIM8 tool automatically prepares your design for functional simulation and, optionally, loads it into the QuickSim II simulator. You need to make a few selections in the functional simulation dialog box to guide the processing of your design. For most designs, you must run PLD_Men2XNF8 before running PLD_FNCSIM8. PLD_Men2XNF8 creates an XNF file from your design schematic. If you want to manually prepare your design for functional simulation from the command line, refer to the “Manual Translation” chapter.

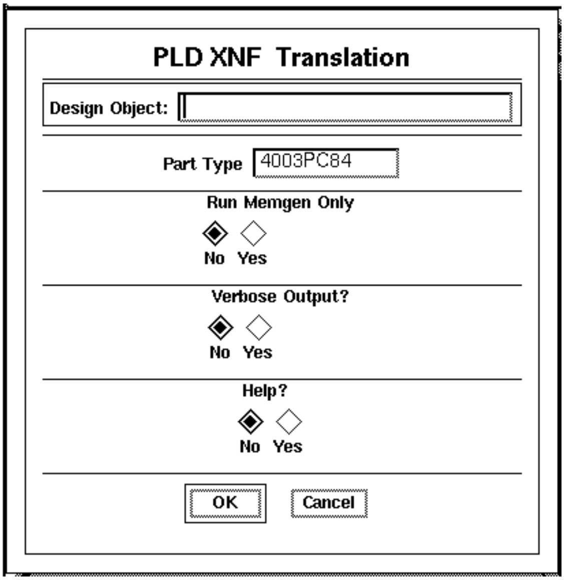
This chapter also provides general information on the QuickSim II, QuickPath, and PLD_DVE tools. For more information on functional simulation, refer to the “Simulation Issues” chapter and to the tutorial chapters.

PLD_Men2XNF8

Note: The PLD_Men2XNF8 tool runs the Men2XNF8 script, which converts your schematic design to an XNF file. For most designs, you must run PLD_Men2XNF8 before running PLD_FNCSIM8. You do not need to run Men2XNF8 first if your design contains only schematic elements. If your design contains any non-schematic elements, including CLB / IOB symbols, X-BLOX symbols, or symbols with an attached FILE property, you must run PLD_Men2XNF8 before PLD_FNCSIM8. If you make a change to your design, you must run PLD_Men2XNF8 again before running PLD_XMake or PLD_FNCSIM8, so that the schematic change is reflected in the netlist.

Note: For EPLD designs, functional simulation is supported only for designs consisting entirely of Xilinx-supplied XC7000 library symbols. Designs containing PLD symbols or custom primitives defined using equation files cannot be functionally simulated. Instead, proceed with design implementation and use the timing simulation procedure to verify your design.

After you capture your design in Design Architect, check and save it before you invoke PLD_Men2XNF8. To invoke PLD_Men2XNF8, perform a data-centered or a tool-centered invocation on the icon. The PLD XNF Translation dialog box shown in the following figure appears on the screen.



PLD XNF Translation

Design Object:

Part Type 4003PC84

Run Memgen Only
◆ No ◇ Yes

Verbose Output?
◆ No ◇ Yes

Help?
◆ No ◇ Yes

Figure 6-1 PLD XNF Translation Dialog Box

Select options from the PLD XNF Translation dialog box as follows:

Design Object

Enter the name of your design object.

Note: This field will not appear if you performed a data-centered invocation of PLD_Men2XNF8 from the Navigator window.

Part Type

Enter the correct Xilinx part type.

Note: The part type that appears in the window is either the default part type or the last part type entered.

Run MemGen Only

Select this option to prevent the creation of an XNF file. When you select this option, Men2XNF8 searches your working directory and runs the MemGen program on any .mem, RAM, or ROM description files that are in the directory. See the *XACT Reference Guide* for more information on the MemGen program.

Verbose Output

Select this option to run PLD_Men2XNF8 in verbose mode. All output messages, including the various programs that are executed, are shown on the screen as well as recorded in the men2xnf8.log file. If this option is not selected, the output messages are still recorded in the men2xnf8.log file.

Help

This option displays the PLD_Men2XNF8 help text, which includes a summary of the program’s syntax and a brief description of the options.

OK or Cancel

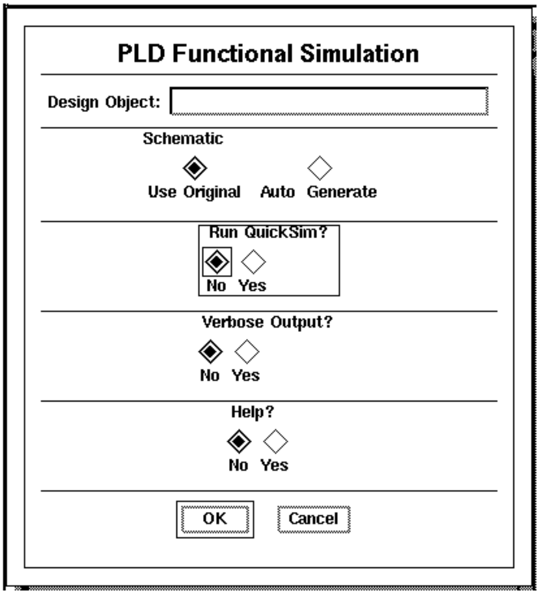
Select **`OK`** to start PLD_Men2XNF8 or select **`Cancel`** to go back to the Design Manager window without running PLD_Men2XNF8.

PLD_FNCSIM8

Note: EPLD designs that contain PLD symbols cannot be functionally simulated.

The PLD_FNCSIM8 tool runs the FNCSIM8 script. FNCSIM8 reads your design data base, creates the simulation viewpoint, and, optionally, executes the Mentor Graphics QuickSim II program. You can make subsequent edits to the schematic using PLD_DA, and then immediately execute QuickSim II without invoking PLD_FNCSIM8 again.

After you capture your design in Design Architect, check and save it before you invoke PLD_FNCSIM8. To invoke PLD_FNCSIM8, perform a data-centered or a tool-centered invocation on the icon. The PLD Functional Simulation dialog box shown in the following figure appears on the screen.



PLD Functional Simulation

Design Object:

Schematic

◆ Use Original ◇ Auto Generate

Run QuickSim?

◆ No ◇ Yes

Verbose Output?

◆ No ◇ Yes

Help?

◆ No ◇ Yes

Figure 6-2 PLD Functional Simulation Dialog Box

Select options from the PLD Functional Simulation dialog box as follows:

Note: The options described below are identical for FPGA and EPLD designs.

Design Object

Enter the name of your design object.

Note: This field does not appear if you performed a data-centered invocation of PLD_FNCSIM8 from the Navigator window.

Schematic

Select either **`Use Original`** or **`Auto Generate`** as follows:

Use Original

This is the default option. Select this option for all EPLD designs and for FPGA designs that do not contain CLB, IOB, or EQN primitives. This option specifies that the original schematic is used for functional simulation. If your design contains symbols that have an attached FILE property or if your design contains X-BLOX primitives, the Gen_Sch8 program is automatically run to create schematics for simulation. If your design contains Xilinx ABEL symbols, ABL2XNF and Gen_Sch8 are automatically run to create schematics for simulation. This option allows you to easily debug and probe your original schematic in QuickSim II.

Auto Generate

Note: This option only applies to FPGA designs.

Select this option for FPGA designs that contain CLB, IOB, or EQN primitives. This option automatically creates a new schematic for functional simulation. CLBs, IOBs, and EQNs are non-schematic elements without an attached FILE property. Since an XNF file does not exist for the CLB/IOB/EQN primitives, a quick place and route is performed on the entire design to create an LCA file. The LCA file is then converted by LCA2XNF to a flattened XNF file. The Gen_Sch8 program is then run to create a completely new flat schematic. Finally, a simulation viewpoint is created and functional simulation can be performed.

If you have many X-BLOX symbols in your design, you may want to select this option to decrease processing time. However, it is important to note that the generated schematic is a single page, flat schematic that does not resemble your original schematic.

Run QuickSim

Select this option to automatically load your design into the QuickSim II program for functional simulation. You can specify simulation options for your design within QuickSim. If you want to set options before invoking the simulator program, select **No** and use the QuickSim II icon to set options. See the QuickSim II section below for more information.

Verbose Output

Select this option to run PLD_FNCSIM8 in verbose mode. All output messages including the various programs that are executed are shown on the screen as well as recorded in the fncsim8.log file. If this option is not selected, the output messages are still recorded in the fncsim8.log file.

Help

This option displays the PLD_FNCSIM8 help text, which includes a summary of the program's syntax and a brief description of the options.

OK or Cancel

Select **OK** to start PLD_FNCSIM8 or select **Cancel** to go back to the Design Manager window without running PLD_FNCSIM8.

Output Files

The following output files are generated by PLD_FNCSIM8:

- **fncsim8.log**: all fncsim8 output messages are displayed on the screen as well as piped to this file. Check this file for error messages.
- **fncsim8.sh**: fncsim8 command file. You can edit this file to add options or customize the command sequence.

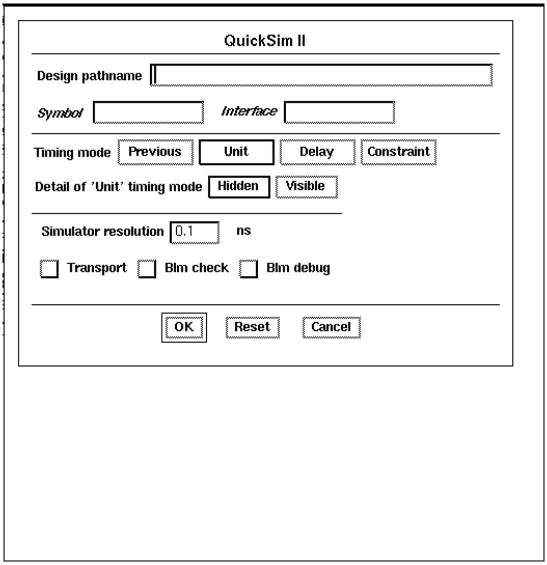
- **men2xnf8.log**: all men2xnf output messages are displayed on the screen as well as piped to this file. Check this file for error messages.
- **men2xnf8.sh**: men2xnf command file. You can edit this file to add options or customize the command sequence.
- **default.dvpt**: default viewpoint file. This file can be used for both functional and timing simulation.

QuickSim II

Note: If you selected the Run QuickSim option in the Functional Simulation dialog box, your design is automatically loaded into the QuickSimII simulator and the dialog box shown below does not appear.

Every design must have a simulation viewpoint before it can be used in QuickSim. The viewpoint defines such things as which simulation model should be used for a primitive.

After preparing your design for functional simulation, you can invoke QuickSim II to simulate your design. To invoke, double-click the left mouse button on the QuickSim II icon. The QuickSim II dialog box shown in the following figure appears on the screen. For more detailed information on the dialog box options, refer to the Mentor Graphics QuickSim II documentation.



QuickSim II

Design pathname

Symbol *Interface*

Timing mode: Unit

Detail of 'Unit' timing mode: Hidden

Simulator resolution 0.1 ns

□ Transport □ Blm check □ Blm debug

Figure 6-3 QuickSim II Dialog Box

Design Pathname

Enter your design directory.

Symbol

This is an advanced option and can be ignored for most designs. Refer to Mentor Graphics documentation for more information.

Interface

This is an advanced option and can be ignored for most designs. Refer to Mentor Graphics documentation for more information.

Timing Mode

Select **`Unit`** for functional simulation.

Select **`Visible`** to display further options for the Unit Timing Mode.

Simulator Resolution

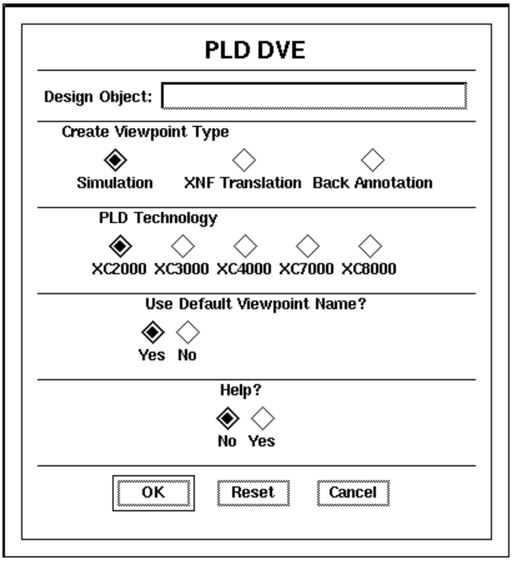
The smallest resolution allowed for Xilinx designs is 0.1 ns.

OK, Reset, or Cancel

Select **`OK`** to start QuickSim II. Select **`Reset`** to reset the dialog box options to the default values. Select **`Cancel`** to go back to the Design Manager window without running QuickSim II.

PLD_DVE

Use PLD_DVE to execute the script that invokes the Mentor Graphics Design Viewpoint Editor (DVE) configured for Xilinx designs. DVE is an interactive application that lets you create Simulation, XNF, or Back Annotation viewpoints for Xilinx designs. To invoke DVE, double-click on the PLD_DVE icon. The dialog box shown in the following figure appears on the screen. For a more detailed description of DVE, refer to the Mentor Graphics documentation.



PLD DVE

Design Object:

Create Viewpoint Type

◆ Simulation ◇ XNF Translation ◇ Back Annotation

PLD Technology

◆ XC2000 ◇ XC3000 ◇ XC4000 ◇ XC7000 ◇ XC8000

Use Default Viewpoint Name?

◆ Yes ◇ No

Help?

◆ No ◇ Yes

Figure 6-4 PLD DVE Dialog Box

Select options from the PLD DVE dialog box as follows:

Design Object

Note: If you performed a data-centered invocation of PLD_DVE, this option will not appear.

Enter the name of your design directory.

Create Viewpoint Type

Simulation

Select this option to create a simulation viewpoint.

XNF Translation

Select this option to create an XNF Translation viewpoint.

Back Annotation

Select this option to create a Back Annotation viewpoint. When you select this option, you are prompted for the Mentor Back Annotation (MBA) file pathname.

PLD Technology

Select the appropriate technology type.

Note: XC8000 is currently not available.

Use Default Viewpoint Name

The default viewpoint is created by the PLD_DVE_SIM script and is used during both functional and timing simulation. Select **`No`** if you do not want to use the default viewpoint name. You are prompted for a new design viewpoint name.

Help

This option displays the PLD_DVE help text, which includes a summary of the program's syntax and a brief description of the options.

OK, Reset, or Cancel

Select **`OK`** to start PLD_DVE. Select **`Reset`** to reset the dialog box options to the default values. Select **`Cancel`** to go back to the Design Manager window without running PLD_DVE.

Mentor Graphics Interface/ Tutorial Guide

Design Implementation

Mentor Graphics Interface/Tutorial Guide — 0401408 01 *Printed in U.S.A.*

Chapter 7

Design Implementation

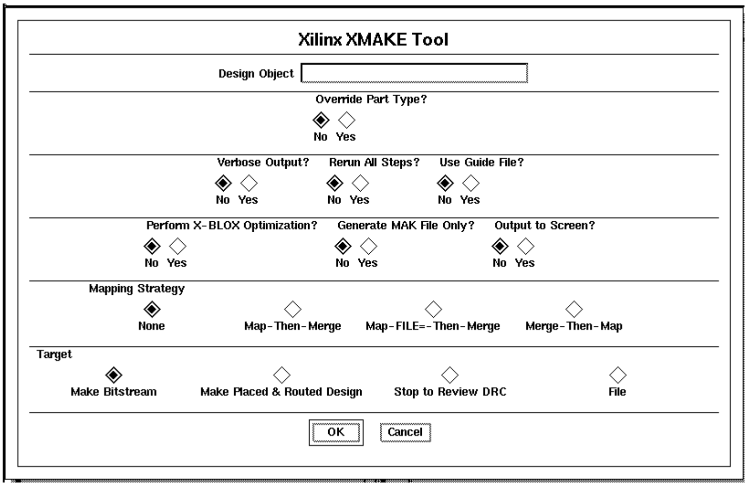
This chapter provides information on implementing FPGA and EPLD designs using the Design Manager tools PLD_XMake and PLD_XEMake, respectively. You must run PLD_Men2XNF8 on all designs before running PLD_XMake or PLD_XEMake. PLD_Men2XNF8 translates your designs to Xilinx Netlist Format (XNF) files. PLD_XMake reads the XNF file and creates logic cell array (LCA) and BIT files. PLD_XEMake reads the XNF file and creates VMH / VMD and Intel HEX files. See the "Functional Simulation Preparation" chapter for information on PLD_Men2XNF8.

The steps performed automatically by PLD_XMake and PLD_XEMake can be executed manually from the UNIX command line. Refer to the "Manual Translation" chapter for more information. For a detailed explanation of XMake and XEMake options, refer to the *XACT Reference Guide* for FPGA designs and to the X*EPLD Reference Guide* for EPLD Designs.

PLD_XMake

Note: Only FPGA designs can be processed with PLD_XMake.

To start PLD_XMake, perform a data-centered or a tool-centered invocation on the icon in the Design Manager window. The Xilinx XMake Tool dialog box shown in the following figure will appear on the screen.



Xilinx XMAKE Tool

Design Object [ ]

Override Part Type? ◆ No ◇ Yes

Verbose Output? ◆ No ◇ Yes

Rerun All Steps? ◆ No ◇ Yes

Use Guide File? ◆ No ◇ Yes

Perform X-BLOX Optimization? ◆ No ◇ Yes

Generate MAK File Only? ◆ No ◇ Yes

Output to Screen? ◆ No ◇ Yes

Mapping Strategy: ◆ None ◇ Map-Then-Merge ◇ Map-FILE=-Then-Merge ◇ Merge-Then-Map

Target: ◆ Make Bitstream ◇ Make Placed & Routed Design ◇ Stop to Review DRC ◇ File

Figure 7-1 XMake Dialog Box

Select dialog box options as follows:

Design Object

Enter the name of your design object.

Note: This field will not appear if you invoked PLD_XMake from your design in the Navigator window.

Override Part Type

The part type entered in the PLD_Men2XNF8 dialog box is inserted into the netlist file that is created by Men2XNF8 and then read by PLD_XMake. To change this part type, select **Yes** for this option and enter a new part type.

Verbose Output

Select this option to run PLD_XMake in verbose mode. All output messages including the various programs that are executed will be shown on the screen as well as piped to the *design*.out file. If this option is not selected, the output messages will still be piped to the *design*.out file.

Rerun All Steps

Use this option to ensure that PLD_XMake reprocesses the entire design, including unchanged submodules from the last time the design was processed.

Use Guide File

Select this option to use a placed and routed design from a prior iteration of PLD_XMake to guide the place and route of a subsequent design iteration after modifications to the design have been made. Enter your guide file name in the file name field that appears when you select this option.

Perform X-BLOX Optimization

Note: This option is only applicable to XC3000A, XC3000L, XC3100A, XC4000, and XC4000A / H designs.

Use this option to force the execution of the X-BLOX program on a design, even on one that does not use X-BLOX symbols, in order to take advantage of the optimization X-BLOX performs.

Generate MAK File Only

Use this option to create a MAK file only, without running the commands in the MAK file to implement the design. Select this option when you want to create a custom MAK file, forcing XMake to generate a script that you can edit.

Output to Screen

Use this option to direct all program output to the screen instead of generating a *design*.out file.

Mapping Strategy

Note: Supports XC2000, XC2000L, XC3000, XC3100, XC3000A / L, XC3100A only.

Use this option to control mapping strategy. It is sometimes preferable to map sections of a design separately before merging the different sections together.

Map-Then-Merge

Select this option to use the ‘map-then-merge’ mapping strategy. XMake automatically selects this option for XNFMAP, regardless of your settings in the option profile.

Map-FILE=-Then-Merge

Select this option to use the ‘map-FILE=-then-merge’ mapping strategy.

Merge-Then-Map

Select this option to use the ‘merge-then-map’ mapping strategy.

Target

By default, PLD_XMake processes a design all the way to a configuration bitstream BIT file, unless you specify a different target. XMake will stop after creating the specified target file.

Make Bitstream

Select this option to create a bitstream file.

Make Placed & Routed Design

Select this option to create an LCA file. The MakeBits program will not be run by PLD_XMake.

Stop to Review DRC

This option sets the target to *design*.xft if you are processing XC4000 devices. PLD_XMake will not run PPR, XDelay, or MakeBits during the process. This option sets the target to *design*.map if you are processing XC3000A / L devices. PLD_XMake will not run PPR, XDelay, or MakeBits during the process. This option sets the target to *design*.map if you are processing XC2000, XC3000, or XC3100 devices. PLD_XMake will not run MAP2LCA, APR, or MakeBits during the process. If you specify a target in the target field, PLD_XMake ignores this option.

File

You will be prompted for a target file name when you select this option. PLD_XMake stops after creating the named file.

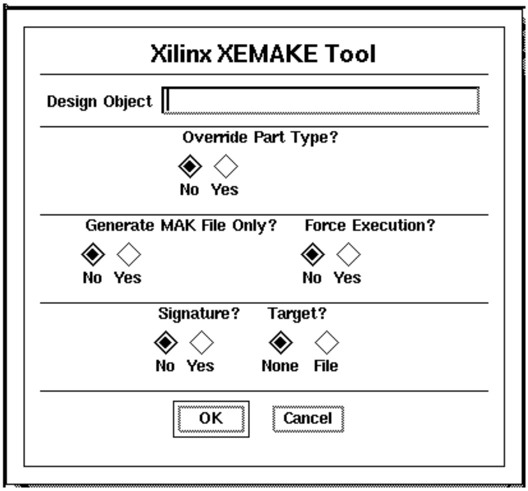
OK or Cancel

Select **OK** to start PLD_XMake or select **Cancel** to go back to the Design Manager window without running PLD_XMake.

PLD_XEMake

Note: Only EPLD designs can be processed with PLD_XEMake.

To start PLD_XEMake, perform a data-centered or a tool-centered invocation on the icon in the Design Manager window. The Xilinx XEMake Tool dialog box shown in the following figure appears on the screen.



Xilinx XEMAKE Tool

Design Object

Override Part Type? ◆ No ◇ Yes

Generate MAK File Only? ◆ No ◇ Yes

Force Execution? ◆ No ◇ Yes

Signature? ◆ No ◇ Yes

Target? ◆ None ◇ File

Figure 7-2 XEMake Dialog Box

Select dialog box options as follows:

Design Object

Enter the name of your design object.

Note: This field will not appear if you invoked PLD_XEMake from your design in the Navigator window.

Override Part Type

The part type entered in the PLD_Men2XNF8 dialog box is inserted into the netlist file that is created by Men2XNF8 and then read by PLD_XEMake. To change this part type, select **Yes** for this option and enter a new part type.

Generate MAK File Only

This option causes PLD_XEMake to create a MAK file only, without continuing with the commands in the MAK file to implement the design. Use this option when you want to create a custom MAK file, forcing PLD_XEMake to generate an initial script that you can edit.

Force Execution

This option forces execution of all design files regardless of file dates, and displays makefiles in the input file list along with the schematic and behavioral design files.

Target

PLD_XEMake will create a VMH file unless you specify a different target. To specify a target, select **File**. You will be prompted for a file name. Type in a file name with one of the following extensions. PLD_XEMake will stop after creating the specified target file.

Table 7-1 Target Specification

File Description	File Extension
Standard database file	.vmh
Database file for XC7272 part only	.vmd
Intel Hex programming file	.prg

Signature

Use this option only if you have specified an Intel Hex file as your target. You will be prompted for a signature to be used as a chip label. The signature can be any unique identifier up to 8 characters in length. A ".a" extension identifies the signature as ASCII.

Output Files

The following output files are generated by PLD_XMake and PLD_XEMake.

Design File

Design.lca is the name of the design file generated by PLD_XMake. PLD_XMake creates an LCA file that is partitioned, placed, and routed by either the APR (XC2000 / XC3000 designs) or PPR (XC3000A / L and XC4000 designs) program.

Design.vmh or *design*.vmd is the name of the partitioned design file generated by PLD_XEMake.

Note: *Design*.vmd is the output file for XC7272 designs only. The *design*.vmh file is generated for all other EPLD designs.

Programming File

Design.bit is the name of the bitstream file, generated by PLD_XMake, that can be downloaded to an FPGA device.

Design.prg is the name of the bitmap file, generated by PLD_XEMake, that can be programmed into an EPLD device.

MAK File

Both PLD_XMake and PLD_XEMake create a MAK file that documents how each design submodule is processed, including the options used by the translation programs. You can examine the MAK file to determine exactly which programs and options were used to process your design.

Report File

Design.rpt is the name of the report file generated by PLD_XMake. It contains the results of the place and route routines. Check this file to make sure there are no unrouted pins or nets.

PLD_XEMake generates several report files. Refer to the *XEPLD Reference Guide* for more information.

Note: The following output files are only generated by PLD_XMake.

Out File

The *design*.out file contains all the text that is echoed to the screen during the processing of your design, including all warnings and error messages as well as the programs run by PLD_XMake. Check this file after running PLD_XMake to ensure your design is error-free.

PRP File

The *design*.prp file is the Design Rule Check (DRC) report file generated by XNFPrep. If XNFPrep finds any errors or warnings, the *design*.out file directs you to examine this file. This file contains a detailed list of all logic trimmed by XNFPrep.

Men2XNF8.log File

This file contains a transcript of the outputs of the programs run by the schematic translation script,Men2XNF8. Check this file for errors if PLD_XMake or PLD_XEMake fails before creating an XNF netlist.

Mentor Graphics Interface/ Tutorial Guide

Timing Simulation Preparation

Printed in U.S.A.

Chapter 8

Timing Simulation Preparation

This chapter explains how to use the PLD_TIMSIM8 tool to process your PLD designs and create the files necessary to perform timing simulation with the QuickSim II simulator. Timing simulation verifies design functionality by using delay information from the LCA file or VMH / VMD file created during design implementation. This delay information must be back-annotated to a schematic before it can be used for timing simulation. If your design contains only schematic elements, back-annotation to the original schematic is possible. If your design contains non-schematic elements, back-annotation to the original schematic is not possible, and a new schematic must be generated before timing simulation is possible.

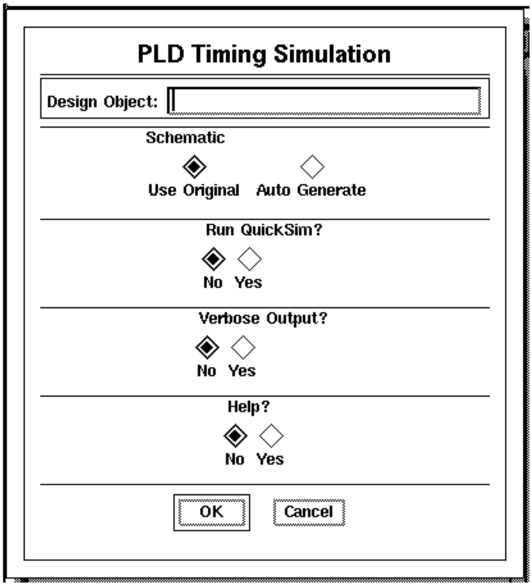
The PLD_TIMSIM8 tool automatically prepares your design for timing simulation and, optionally, loads it into the QuickSim II simulator. You need to make a few selections in the simulation dialog box to guide the processing of your design. If you want to manually prepare your design for timing simulation from the command line, refer to the “Manual Translation” chapter.

For more information on timing simulation, refer to the “Simulation Issues” chapter and to the tutorial chapters.

PLD_TIMSIM8

The PLD_TIMSIM8 tool runs the TIMSIM8 script. TIMSIM8 automatically prepares your design for timing simulation by reading the LCA or VMH / VMD file and back-annotating the delay information to the original schematic or to a newly generated schematic.

To start PLD_TIMSIM8, perform a data-centered or a tool-centered invocation on the icon. The PLD Timing Simulation dialog box shown in the following figure appears on the screen:



PLD Timing Simulation

Design Object:

Schematic

◆ Use Original ◇ Auto Generate

Run QuickSim?

◆ No ◇ Yes

Verbose Output?

◆ No ◇ Yes

Help?

◆ No ◇ Yes

Figure 8-1 PLD Timing Simulation Dialog Box

Select options from the PLD Timing Simulation dialog box as follows:

Note: The options described below are identical for FPGA and EPLD designs.

Design Object

Enter the name of your design object.

Note: This field does not appear if you performed a data-centered invocation of PLD_TIMSIM8 from the Navigator window.

Schematic

Select either **`Use Original`** or **`Auto Generate`** as follows:

Use Original

Note: This option is only applicable to FPGA designs.

Select this option for FPGA designs that contain only schematic elements. This option specifies that the delay information in the LCA file is back-annotated to the original schematic. If your design contains any of the symbols listed below for the Auto Generate option, do not select the Use Original option.

Auto Generate

Select this option for all EPLD designs and for FPGA designs that contain CLB, IOB, EQN, X-BLOX, Xilinx ABEL, and MemGen symbols. The delay information in the LCA file or VMH / VMD file must be back-annotated to a schematic before it can be used for timing simulation. For the designs listed above, back-annotation to the original schematic is not possible since there is no direct correlation between the original schematic and the file created during implementation. A new schematic must be generated from the information in the LCA or VMH / VMD file. A simulation viewpoint is then created from the new schematic and timing simulation can be performed.

Run QuickSim

Select this option to automatically load your design into the QuickSim II program for timing simulation. You can specify simulation options for your design within QuickSim. If you want to set options before invoking the simulator program, select **No** and use the QuickSim II icon to set options. See the QuickSim II section below for more information.

Verbose Output

Select this option to run PLD_TIMSIM8 in verbose mode. All output messages, including the various programs that are executed, are shown on the screen as well as recorded in the timsim8.log file. If this option is not selected, the output messages are still recorded in the timsim8.log file.

Help

This option displays the PLD_TIMSIM8 help text, which includes a summary of the program's syntax and a brief description of the options.

OK or Cancel

Select **OK** to start PLD_TIMSIM8 or select **Cancel** to go back to the Design Manager window without running PLD_TIMSIM8.

Output Files

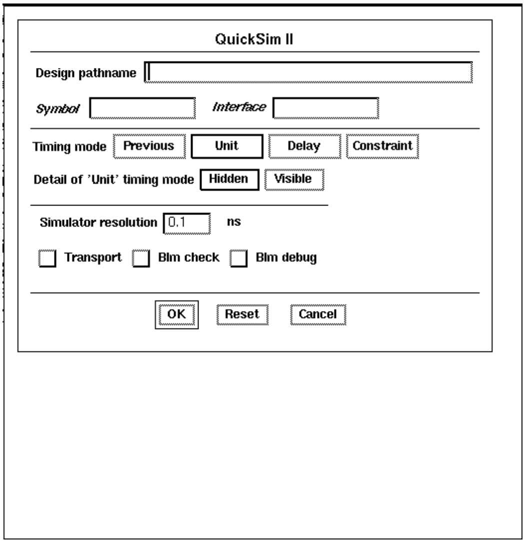
The following output files are generated by PLD_TIMSIM8:

- **timsim8.log**: all timsim8 output messages are displayed on the screen as well as piped to this file. Check this file for error messages.
- **timsim8.sh**: timsim8 command file. You can edit this file to add options or customize the command sequence.

QuickSim II

Note: If you selected the Run QuickSim option in the Timing Simulation dialog box, your design is automatically loaded into the QuickSim II simulator and the dialog box shown below does not appear.

To start QuickSim II, double-click the left mouse button on the QuickSim II icon. The QuickSim II dialog box shown in the following figure appears on the screen. For more detailed information on the dialog box options, refer to the Mentor Graphics documentation.



Design pathname

Symbol *Interface*

Timing mode Previous [✓ Unit] Delay Constraint

Detail of 'Unit' timing mode [✓ Hidden] Visible

Simulator resolution 0.1 ns

□ Transport □ Blm check □ Blm debug

Figure 8-2 QuickSim II Dialog Box

Design Pathname

Enter your design directory.

Symbol

This is an advanced option and can be ignored for most designs. Refer to Mentor Graphics documentation for more information.

Interface

This is an advanced option and can be ignored for most designs. Refer to the Mentor Graphics documentation for more information.

Timing Mode

Select **Delay** in the Timing Mode row to specify that actual delays from the implemented design are used for timing simulation.

Select **Visible** to display further options for the Delay Timing Mode.

Simulator Resolution

The smallest resolution allowed for Xilinx designs is 0.1 ns.

OK, Reset, or Cancel

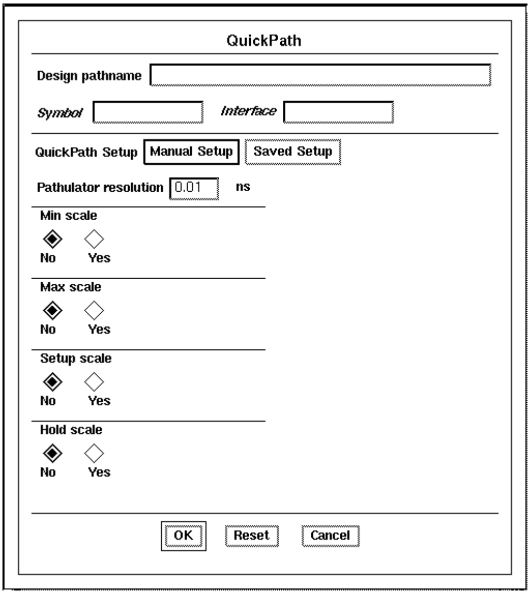
Select **OK** to start QuickSim II. Select **Reset** to reset the dialog box options to the default values. Select **Cancel** to go back to the Design Manager window without running QuickSim II.

QuickPath

Note: Running the QuickPath application on PLD designs is optional.

Use the Mentor Graphics QuickPath tool to perform static and slack timing analysis on schematic designs that have been prepared for timing simulation. This tool enables you to identify critical paths and evaluate modifications that can improve your circuit's performance. Use the timing analysis tool to determine possible changes to a circuit so that you can optimize its performance. Refer to the Mentor Graphics documentation for more information.

To start QuickPath, double-click on the QuickPath icon in the Design Manager window. The dialog box shown in the following figure appears on the screen. For more information on the dialog box options, refer to the Mentor Graphics documentation.



QuickPath

Design pathname

Symbol

Interface

QuickPath Setup: Manual Setup | Saved Setup

Pathulator resolution 0.01 ns

Min scale: ◆ No ◇ Yes

Max scale: ◆ No ◇ Yes

Setup scale: ◆ No ◇ Yes

Hold scale: ◆ No ◇ Yes

Figure 8-3 QuickPath Dialog Box

Mentor Graphics Interface/ Tutorial Guide

Simulation Issues

Chapter 9

Simulation Issues

This chapter provides important information you need to consider when using the QuickSim II program to simulate your PLD designs.

Simulation Models

Most Xilinx simulation models are built with the Mentor Graphics QuickPart tables. Flip-flops and memory elements are modeled with QuickPart tables and behavioral language models, while gates are modeled with QuickPart tables. All delay information is passed to Xilinx components via the routed XNF file.

Analyzing Nets from the Schematic

This section describes how to select and analyze nets within the QuickSim II simulator.

You can probe nets in QuickSim II by opening a schematic sheet and selecting a net. To trace the selected signal:

1. Select the (schematic view) **`Add`** → **`Traces`** → **`Selected`** menu path.

   A Trace window is created with the selected signals.

2. You can List and Monitor selected nets by selecting the (schematic view) **`Add`** → **`Lists`** → **`Selected`** and (schematic view) **`Add`** → **`Monitors`** → **`Selected`** menu items.

After you have set up a list of signals, you can save the list in a “do file” to use in future QuickSim sessions. Refer to the QuickSim II manuals from Mentor Graphics for detailed information on using the simulator and creating do files.

FPGA Devices

Global Reset and 3-State Signals

Before you simulate a design, you must force the globalresetb (XC2000 and XC3000 designs) or the globalsetreset (XC4000 designs); otherwise, the flip-flops and latches do not function correctly.

1. Select your design directory icon in the Navigator window and select **`Right Mouse Button → Open → QuickSimII`** to enter the QuickSim II simulator.
2. Select the **`File → Open Sheet`** menu item to display the Design Architect schematic.
3. Select the **`Add Force`** menu from the QuickSim II Stimulus palette.
4. Fill in the dialog box with the //globalresetb signal name, 0 for the first time, and 0 for the first value; <n> for the second time, and 1 for the second value.

   <n> should not be less than the minimum reset width for the given speed grade of the design specified in the *The Programmable Logic Data Book* for the XC2000 and XC3000 families. The reset width emulates a power–on reset at the beginning of simulation.

   Globalresetb is now forced High at <n> ns. If you want to reset the flip-flops after <n> ns, toggle the globalresetb Low and High for the necessary pulse width specified in *The Programmable Logic Data Book*. Setting the //globalresetb signal in simulation is similar to toggling the Reset pin on an XC2000 or XC3000 device after configuration.

The previous procedure is slightly different for XC4000 IOBs and 3-state I/O pins.

To set XC4000 IOB flip-flops:

1. Set the IOB flip-flops High or Low on power-up by using the INIT property on the IOB flip-flops.
2. To activate the signal and begin simulation, set globalsetreset by selecting the **`Add Force`** menu item from the QuickSim II Stimulus palette.

3. Fill in the dialog box with the //globalsetreset signal name, 0 for the first time and 1 for the first value; <n> for the second time and 0 for the second value.

   <n> is the specified minimum reset pulse width for the given speed grade part of the design, specified in *The Programmable Logic Data Book*.

XC4000 parts have a global input state to make all output pins 3-state, which allows the isolation of the XC4000 part in board test. To simulate the global 3-state signal, force the signal named //globalthreestate High using the Add Force command. Forcing the signal High holds all chip I/Os in a high–Z (3–state) state until //globalthreestate is forced to zero.

XC4000 Simulation Exceptions

This section describes issues that you need to be aware of when simulating XC4000 ROMs.

For full-timing simulation, PPR reads in the INIT property on the ROM and translates it into the appropriate logic gates. Because PPR hard codes the initial value into the LCA file, you cannot change the initial value after running it. Unlike RAMs, ROMs can have pre-programmed data in the Xilinx part.

EPLD Devices

Using PRLD for Initialization

The PRLD (preload) signal is an input to your EPLD design that does not appear on your schematic; it is included in the models of registered components and is automatically added to the functional and timing models by FNCSIM8 and TIMSIM8, respectively. You must include the PRLD signal in your do file to ensure that the registers in your design are initialized properly.

Before applying simulation stimuli, you must initialize the device by pulsing PRLD High for at least 1 time unit (0.1ns). Before initialization, all registers are in an unknown state (U) which usually prevents any meaningful simulation results. PRLD simulates the Master Reset signal (or power-on-reset) of the EPLD device and forces all registers to a predefined state.

All input signals to the device should be set to a known logic state before PRLD is returned Low, otherwise some internal nodes may become trapped in an unknown state. You must return PRLD to a Low state before the design will respond properly to input stimulus.

For functional simulation, all registered components initialize to the state defined in the *XACT Libraries Guide*. During implementation, the XEMake program might alter the initial states of register to take optimal advantage of device resources unless you inhibit Preload optimization. Timing simulation will exhibit the actual register preload values implemented by the software. You can control the preset state of XEPLD registers in PLD components, so they are forced either High or Low, by using the *"output.PRLD"* equation in PLUSASM. See the *XEPLD Reference Manual* for more information on the PRLD extension.

You can use the PRELOAD_OPT global attribute to determine whether the preload value can be changed for parts of the design to allow logic to fit as efficiently as possible into device resources (ON), or the logic is mapped to device resources according to the established preload values (OFF). PRLD equations in your PLD components are only effective if PRELOAD_OPT is turned OFF. For more information, see the "EPLD Design Issues" chapter.

Under some conditions, the first simulation cycle after PRLD is brought Low produces setup and hold violations. This is due to the asynchronous nature of the PRLD signal. The resulting warnings are usually not indicative of a circuit problem.

You should analyze your design for any potential initial-state problems that could result when the device comes out of the power-on-reset or the external Master Reset (MR). For example, if the device recovers from its reset cycle and becomes operational coincident with the rising edge of a free-running clock input, not all registers in the device might respond to this first clock cycle. As a result, an invalid internal state might occur. Such situations are not detectable during simulation.

See the "XEPLD Tutorial" chapter for an example of how to initialize the device during simulation.

Forcing PRLD

Before you simulate a design, you must force the PRLD, otherwise, the flip-flops and latches do not function correctly. Perform the following steps to force the PRLD:

1. Select your design directory icon in the Navigator window and select **`Right Mouse Button`** → **`Open`** → **`QuickSimII`** to enter the QuickSim II simulator.
2. Select the **`File`** → **`Open Sheet`** menu item to display the Design Architect schematic.
3. Select the **`Add Force`** menu from the QuickSim II Stimulus palette.
4. Fill in the dialog box with the //PRLD signal name, 0 for the first time, and 1 for the first value; any number greater than 0 for the second time, and 0 for the second value.

XC7000 Simulation Exceptions

For EPLD designs, functional simulation is supported only for designs consisting entirely of Xilinx-supplied XC7000 library symbols. Designs containing PLD symbols or custom primitives defined using equation files cannot be functionally simulated. Use the timing simulation procedure to verify your design.

Mentor Graphics Interface/ Tutorial Guide

Manual Translation

Chapter 10

Manual Translation

This chapter describes how to prepare your designs for functional and timing simulation from the UNIX command line. A brief description of the design implementation flow is also provided.

The first half of the chapter includes the applicable command sequence for various types of designs. Flow diagrams and command summaries are provided to guide you through the processing of your design. The second half of the chapter provides descriptions of the various programs including syntax, variables, and options.

There are three scripts written in the System V Bourne Shell:

- Men2XNF8
- FNCSIM8
- TIMSIM8

These scripts run the various programs in the command sequences provided in the first part of the chapter. You can either enter each program individually at the command line or you can automate the process by running the appropriate script. These three scripts are described in the second half of the chapter.

Use commands exactly as they are shown below for each type of design. The input and output files are given extended names to help track which files to use for later commands and to avoid overwriting any important files. Syntax conventions are shown in the following table:

Table 10-1 Syntax Conventions

italic font	Variables that you replace in syntax statements.
[ ]	Denote optional items or parameters. However, in bus specifications, such as bus [7:0], they are required.
{ }	Enclose a list of items from which you must choose one or more.
|	Separates items in a list of choices.

Functional Simulation

FPGA and EPLD Designs with Only Schematic Elements

Note: The following functional simulation flow is the only one applicable to EPLD designs. For EPLD designs, functional simulation is supported only for designs consisting entirely of Xilinx-supplied XC7000 library symbols. Designs containing PLD symbols or custom primitives defined using equation files cannot be functionally simulated. Use the timing simulation procedure to verify your design.

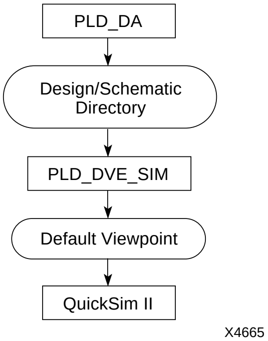


Figure 10-1 Schematic Only (Functional Simulation)

Use the following command sequence on designs that contain only schematic elements. For FPGA designs, do not use this procedure if your schematic contains CLB/IOB, X-BLOX, MemGen, or Xilinx ABEL symbols, or symbols with an attached FILE property. For EPLD designs, do not use this procedure if your schematic contains PLD symbols with an attached PLD=*filename* property. These commands prepare your design for functional simulation.

```
pld_dve_sim design tech_type
quicksim design
```

FPGA Designs with Schematic and CLB/IOB/EQN Elements

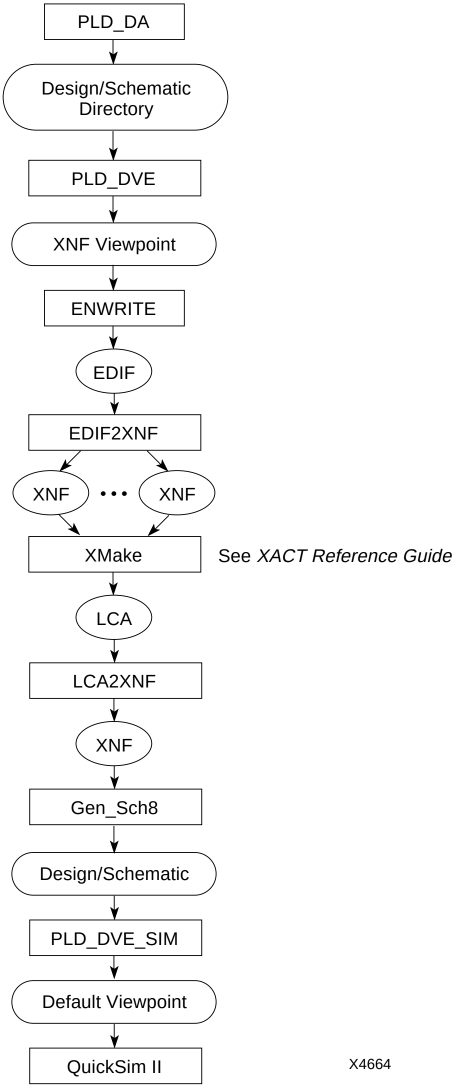


Figure 10-2 Schematic & CLB/IOB/EQN (Functional Simulation)

Use the following command sequence on designs that contain schematic elements and CLB/IOB/EQN primitives. CLBs, IOBs, and EQNs are non-schematic elements without an attached FILE property. Since an XNF file does not exist for the CLB/IOB/EQN primitives, a quick place and route is performed on the entire design to create an LCA file. The LCA file is then converted by LCA2XNF to a flattened XNF file. The Gen_Sch8 program is run to create a completely new flat schematic. A simulation viewpoint is then created and functional simulation can be performed.

```
pld_dve design tech_type

rm design.edif

enwrite design/xnf

     -rcf $LCA/data/enwrite.config

     -wef design.edif

edif2xnf design.edif -p part_type

xnfmerge design.xnf
```

At this point, you must create an LCA file by running the appropriate programs for your design. Refer to the *XACT Development System Reference Guide* for the correct procedure. Once you have created an LCA file, execute the remaining steps as follows:

```
lca2xnf -g design design_fnc

gen_sch8 design_fnc.xnf

pld_dve_sim design_fnc tech_type

quicksim design_fnc
```

FPGA Designs with Schematic and X-BLOX Elements

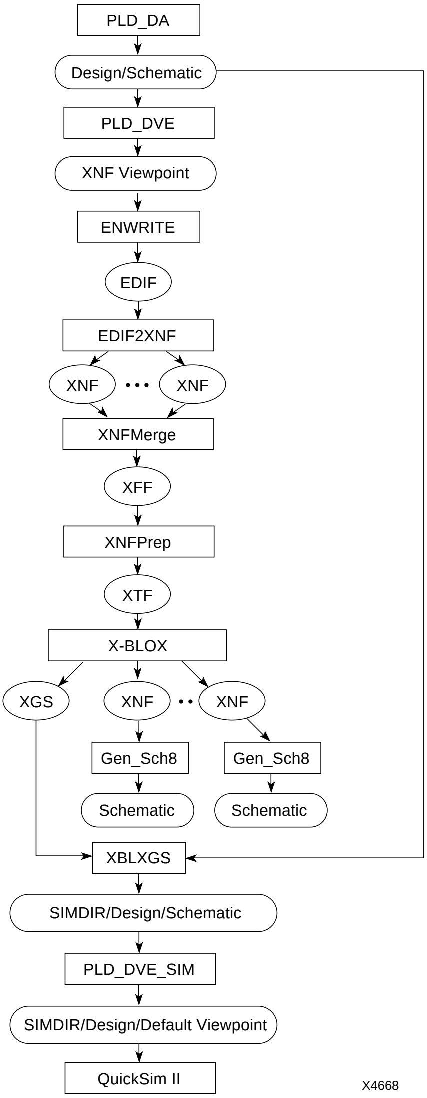


Figure 10-3 Schematic and X-BLOX (Functional Simulation)

Use the following command sequence on designs that contain schematic elements and X-BLOX symbols. X-BLOX symbols are non-schematic elements without an attached FILE property. The X-BLOX program is run to create the necessary XNF files needed to generate a new schematic for simulation.

ENWRITE reads the XNF viewpoint created by PLD_DVE and produces an EDIF file. EDIF2XNF converts the EDIF file to an XNF file. One XNF file is produced for every block in the schematic. XNFMerge converts the XNF files produced by EDIF2XNF into a flattened file with a .XFF extension. XNFPrep checks this file for errors, trims unused logic, and writes a new file with a .XTF extension. X-BLOX is then run on the .XTF file. X-BLOX processes the X-BLOX symbols and generates the appropriate logic. The XNF files produced by X-BLOX are written to the SIMDIR directory under the working directory. X-BLOX also produces a file with a .XGS extension that describes how the simulation schematic should be drawn. XBLXGS reads the .XGS file and the original schematic and produces a new top-level schematic in the SIMDIR directory. Gen_Sch8 creates the simulation schematics from the XNF files produced by X-BLOX. Finally, PLD_DVE_SIM is run to generate a simulation viewpoint.

```
pld_dve design tech_type

rm design.edif

enwrite design/xnf

     -rcf $LCA/data/enwrite.config

     -wef design.edif

edif2xnf design.edif -p part_type

xnfmerge design.xnf

xnfprep design.xff

rm -r simdir

mkdir simdir

xblox design.xtf simdir=simdir sim=xnf

xblxgs design simdir/design.xgs -w -d simdir
```

Note: The “#” symbol in the following command is a variable and represents the number of the bsm file.

```
gen_sch8 simdir/bsm#.xnf -w
pld_dve_sim simdir/design tech_type
quicksim simdir/design
```

FPGA Designs with Schematic Elements and Elements with the FILE Property

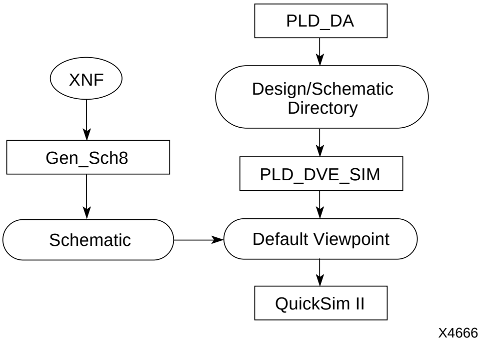


Figure 10-4 Schematic and FILE Property (Functional Simulation)

Use the following command sequence on designs that contain Xilinx ABEL, MemGen, or other symbols with an attached FILE property. The FILE property indicates that the symbol’s logic is represented by an XNF file instead of an underlying schematic. Gen_Sch8 must be run on the XNF files to create a simulation schematic.

```
pld_dve design tech_type
rm design.edif
enwrite design/xnf
     -rcf $LCA/data/enwrite.config
     -wef design.edif
edif2xnf design.edif -p part_type
```

Note: The *subdesign*.xnf file in the following command is the lower level XNF file represented by Xilinx ABEL, MemGen, or other symbols with an attached FILE property.

```
gen_sch8 subdesign.xnf

pld_dve_sim design tech_type

quicksim design
```

FPGA Designs with Schematic Elements, Elements with the FILE Property, and X-BLOX Elements

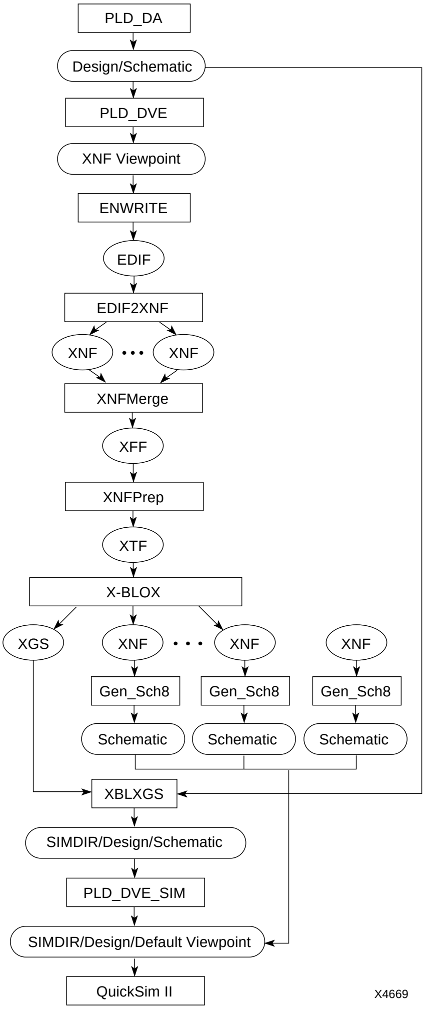


Figure 10-5 Schematic, FILE Property, and X-BLOX (Functional Simulation)

Use the following command sequence on designs that contain schematic elements, elements with the FILE property, and X-BLOX elements. Refer to the two previous design flows for a summary of the commands.

```
pld_dve design tech_type

rm design.edif

enwrite design/xnf

     -rcf $LCA/data/enwrite.config

     -wef design.edif

edif2xnf design.edif

xnfmerge design.xnf

xnfprep design.xff

rm -r simdir

mkdir simdir

xblox design.xtf simdir=simdir sim=xnf

xblxgs design simdir/design.xgs -w -d simdir

gen_sch8 simdir/bsm#.xnf -w
```

Note: The *design*.xnf file in the following command is the sub-level XNF file represented by Xilinx ABEL, MemGen, or other symbols with an attached FILE property.

```
gen_sch8 design.xnf

pld_dve_sim simdir/design tech_type

quicksim simdir/design
```

Design Implementation

This section gives you a brief summary of how to generate implemented FPGA and EPLD designs. For detailed information, consult the *XACT Reference Guide*.

FPGA Designs

To translate an XNF file into an LCA file, use the Xilinx implementation tools. The following figure illustrates the implementation design flow. Note that you must implement your design before you can create a timing simulation file.

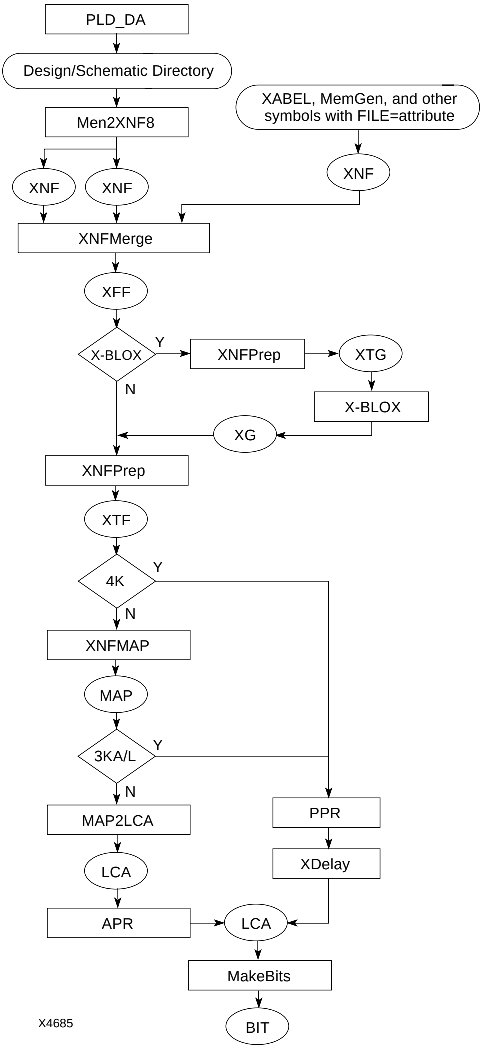


Figure 10-6 FPGA Implementation

EPLD Designs

Run FITNET on your schematic to implement your EPLD design. If you have PLD components or custom primitives in your design, you must first assemble each equation file using PLUSASM before running FITNET. You must implement your design before generating a timing simulation file. To generate a programming file in Intel HEX format (PRG file), use the MakePRG program. Refer to the *XEPLD Reference Guide* for details.

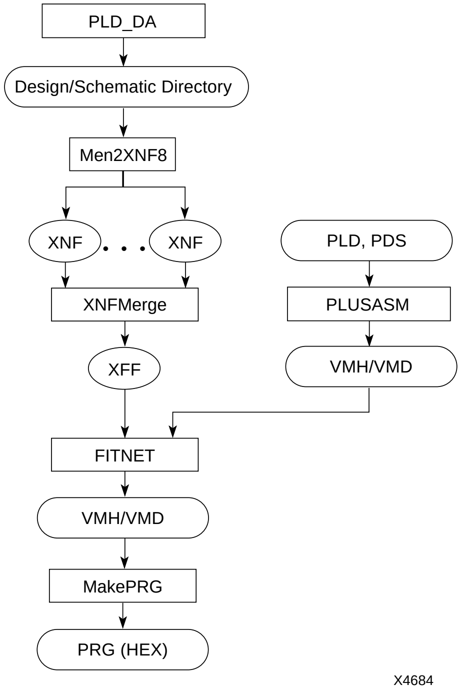


Figure 10-7 EPLD Implementation

Timing Simulation

FPGA Designs with Schematic Elements Only

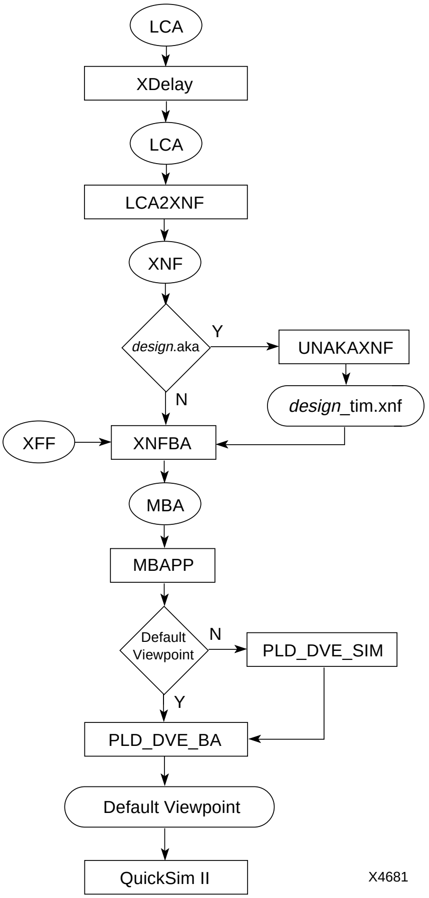


Figure 10-8 Schematic Only (Timing Simulation)

Use the following command sequence on FPGA designs that contain only schematic elements. Do not use this procedure if your design contains CLB/IOB, X-BLOX, MemGen, or Xilinx ABEL symbols, or symbols with an attached FILE property. These commands prepare your design for timing simulation.

Input begins with an LCA file with net delays. XDelay must be run on XC3000A, XC3000L, XC3100A, and XC4000 designs to add these delays. LCA2XNF converts your routed design back to an XNF file. When running LCA2XNF, you must specify an output name that is different from the original XNF file name to avoid overwriting the original *design*.xnf file. UNAKAXNF reads the *design*.aka file, if it exists, that was produced by MAP2LCA and restores the net names that were aliased. XNFBA reads the routed XNF file and the flat pre-routed file created by XNFMerge. It creates a Mentor Back-Annotation (MBA) file. MBAPP is then run to prevent back-annotation ambiguity. The MBA file is read by PLD_DVE_BA to back-annotate the delays to the original default viewpoint. QuickSim II then uses the default viewpoint for timing simulation. The original default viewpoint must exist for back-annotation. You may need to create a default viewpoint with PLD_DVE_SIM.

The first step in the following command sequence only needs to be run on XC3000A, XC3000L, XC3100A, and XC4000 designs.

```
xdelay design.lca -d -w

lca2xnf -g design.lca design_tim.xnf
```

The next two steps are only necessary if a *design*.aka file exists.

```
unakaxnf -o design_timaka design_tim.xnf

mv design_timaka.xnf design_tim.xnf

xnfba design.xff design_tim.xnf -m

nawk -f $LCA/com/{sparc/hppa/apollo}/mbapp.nawk/
   mbafile=design_tim.mba > design_tim.mbb

mv -f design_tim.mbb design_tim.mba

pld_dve_ba design_tim.mba

quicksim design_tim -tim typ -consm messages
```

FPGA Designs with Non-schematic Elements

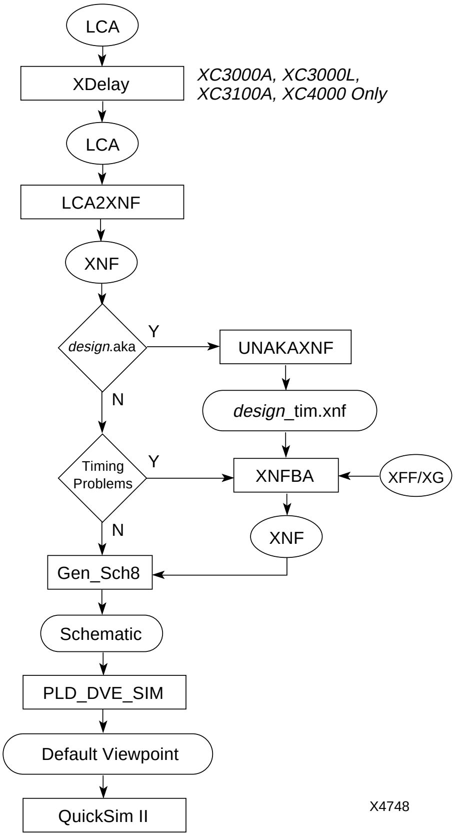


Figure 10-9 FPGA Non-schematic (Timing Simulation)

Use the following command sequence on FPGA designs that contain non-schematic elements. These commands prepare your design for timing simulation.

Input begins with an LCA file with net delays. XDelay must be run on XC3000A, XC3000L, XC3100A, and XC4000 designs to add these delays. LCA2XNF converts your routed design back to an XNF file. When running LCA2XNF, you must specify an output name that is different from the original XNF file name to avoid overwriting the original *design*.xnf file. UNAKAXNF reads the *design*.aka file, if it exists, that was produced by MAP2LCA and restores the net names that were aliased. It is not necessary to run XNFBA on the first pass. If there are no timing problems after the first pass, it is not necessary to run XNFBA. For the first pass, the output of LCA2XNF should be processed by Gen_Sch8. Gen_Sch8 creates a simulation schematic and PLD_DVE_SIM generates a simulation viewpoint.

If XNFBA is run, it creates an XNF file that is read by Gen_Sch8. PLD_DVE_SIM is run to generate a simulation viewpoint.

The first step in the following command sequence only needs to be run on XC3000A, XC3000L, XC3100A, and XC4000 designs.

```
xdelay design.lca -d -w
```

```
lca2xnf -g design.lca design_tim.xnf
```

The next two steps are only necessary if a *design*.aka file exists.

```
unakaxnf -o design_timaka
```

```
mv design_timaka.xnf design_tim.xnf
```

Do not run XNFBA on the first pass. If there are no timing problems, it is not necessary to run XNFBA. XNFBA without the -o option writes output to xnfba.xnf.

```
xnfba {design.xff | design.xg} design_tim.xnf -o
design_tim
```

```
gen_sch8 design_tim.xnf
```

```
pld_dve_sim design_tim
```

```
quicksim design_tim -tim typ -consm messages
```

EPLD Designs

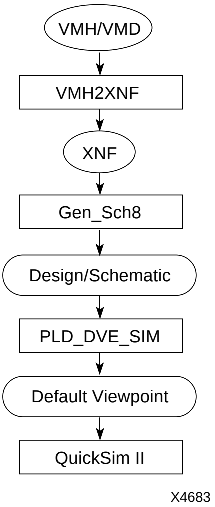


Figure 10-10 EPLD Designs (Timing Simulation)

Use the following command sequence on EPLD designs. These commands prepare your design for timing simulation.

```
vmh2xnf -n design.vmh -o design_tim

gen_sch8 design_tim.xnf

pld_dve_sim design_tim tech_type

quicksim design_tim -tim typ -consm messages
```

Program Summary

Note: When you enter a part type using the -p option, make sure it begins with a 2, 3, 4, or a 7.

EDIF2XNF

The EDIF2XNF program translates an EDIF file into one or more XNF files. Although EDIF2XNF accepts different variations of EDIF files, this section is limited to EDIF files created with the Mentor Graphics ENWRITE program.

Syntax

The EDIF2XNF command syntax is:

`edif2xnf` ***input_file*****`.edif [-`** ***options*****`]`**

The -p option for part type is required.

Variables

input_file.edif is the name of the EDIF input design file that ENWRITE creates. You must specify the .edif extension. The file is created by running ENWRITE on the XNF viewpoint for your design. This viewpoint is created by running PLD_DVE on the EDDM files containing your schematic data created by PLD_DA.

Options

-f, -w Force Overwrite

When the -f option is specified, EDIF2XNF automatically overwrites any existing XNF files. The -f option exists for consistency with other Xilinx programs that use the -w option.

-h Help Text

The -h option displays the EDIF2XNF help text, which includes a summary of the program’s syntax and a brief description of the options.

-l *library_path* Specify EDIF Primitive Library

The -l option defines a search path for the EDIF primitive library, which overrides the $XACT/data/edif* search path (where **`*`** is used as a wild card), if it is set.

You must have the $XACT environment variable set or use the -l option to specify the library path.

-n Do Not Flatten the Design

The -n option causes the design to remain unflattened in the XNF file. By default, EDIF2XNF flattens the design.

-ni Do Not Inherit Properties

The -ni option prevents properties at higher levels from being passed to lower levels of the design. When flattening a design, properties are usually passed down through the design hierarchy and override like properties on subcell instances.

Note: Do not use this option if there are Xilinx soft macros in your design.

-noio Do Not Create EXT Records in XNF File

The -noio option suppresses EXT records in the output XNF file if ports are found in the input EDIF file.

-ns Do Not Skip Unrecognized EDIF Records

EDIF2XNF skips records it does not recognize. The -ns option prevents EDIF2XNF from skipping such records. This option is provided as a debugging tool for use with EDIF formats that are not usually read by EDIF2XNF.

-od *output_directory* Specify Alternate Output Directory

This option specifies an alternate directory for the XNF output files to be placed in that overrides the directory specified when the Xilinx Design Kit is used.

-of Specify Alternate Output File Name

This option specifies an alternate XNF output file name of the top-level structure that overrides the name specified in the EDIF file.

-p *part_type* Specify Part Type

The -p option specifies the Xilinx part type for your design. You can also use this option to override the Part property attached to the EDIF design record in the EDIF file; you must specify one or the other. The part type consists of two parts. The first part is the part name and package type. The second part (optional) is the speed grade. For example:

4005PG156-6

The default part type for EDIF2XNF is 4005PG156; the default speed grade is -6. Do not include any spaces between the part type and the speed grade.

-s Skip Checks

The -s option causes EDIF2XNF to skip the checking of primitives and their ports. Usually, EDIF2XNF checks all primitives to ensure they are known and that all ports for those primitives are known.

-x Flatten Xilinx Soft Macros Only

The -x option flattens only Xilinx soft macros. User-defined blocks in the design are then put into separate XNF files. This option overrides the -n no-flatten option.

-nt Omit Timing

The -nt option omits $Xilinx_add_delays call in the template stimulus file.

-m *map_file* Specify Name of Map File

The -m option specifies the name of a map file that can be used to map foreign EDIF cell primitives into Xilinx ones. $XACT/data/edif*/v*.map is used if it exists, where * is used as a wildcard for 2000, 3000, and 4000.

-v *verilog_map* Generate Verilog Netlist

The -v option generates a Verilog netlist, instead of an XNF file. The map file specifies port order and only accepts EDIF files from XNF to EDIF. Map files reside in $XACT/data/verilog*/v*.map, where * is used as a wildcard for 2000, 3000, and 4000.

ENWRITE

ENWRITE converts an XNF viewpoint to an EDIF file that is read by the EDIF2XNF program. Refer to the Mentor Graphics *EDIF Netlist User's and Reference Manual* for detailed information on ENWRITE.

Syntax

```
enwrite design_name/xnf [-rcf config_file]
      [-wef edif_file]
```

Variables

design_name / xnf specifies the pathname to the component and the XNF viewpoint to be used in creating the EDIF file.

Refer to the Mentor Graphics *EDIF Netlist User's and Reference Manual* for detailed information on -rcf *config_file* and -wef *edif_file.*

Warning: You must specify the XNF viewpoint for the component; otherwise, ENWRITE uses the generic Mentor Graphics default viewpoint for the component and produces unexpected results.

Options

-rcf Read Configuration File

This option instructs ENWRITE to read configuration directives from the specified ASCII configuration file.

-wef edif_fileWrite EDIF File

This option instructs ENWRITE to write an EDIF file of the specified name. You must specify a file name if you use the -wef option.

Note: ENWRITE fails if an EDIF file of the same name already exists, so always check for and remove existing EDIF files before running ENWRITE.

FNCSIM8

FNCSIM8 prepares a Mentor design for QuickSim II functional simulation. It invokes Gen_Sch8, PLD_DVE_SIM, and QuickSim II as needed.

Syntax

```
fncsim8 design {-o | -g}
     [-q -verbose -help]
```

Variables

design is the name of your Mentor design directory.

Options

-o **Use original schematic for simulation**

-g **Generate a new schematic for simulation**

-q **Run QuickSim II**

-verbose **Run FNCSIM8 in verbose mode**

-help **Display FNCSIM8 syntax**

Gen_Sch8

This program creates a new schematic composed of only schematic elements that can be used for functional simulation.

Syntax

```
gen_sch8 [options]
design[.xnf|.xas|.xff|.xtf|.xg]
```

Variables

design.extension is the input xnf, xas, xff, xtf, or xg file

Options

-a aka_file[.aka]

-o output_filename

-s starting sheet_number

-w Silent overwrite

LCA2XNF

LCA2XNF converts a routed design to an XNF file that may be used for functional simulation. Refer to the *XACT Reference Guide* for detailed information on LCA2XNF.

Warning: You should always specify a different output file name so that the original (unplaced, unrouted) XNF file is not overwritten.

Men2XNF8

The Men2XNF8 script translates your design into a Xilinx netlist file (XNF). Men2XNF8 runs PLD_DVE, ENWRITE, EDIF2XNF, and MemGen as needed.

Syntax

```
men2xnf8 design -p part_type
     [-m -verbose -help]
```

Variables

design is the name of your Mentor design directory.

Options

-p Xilinx part type

-m Run only MemGen on .mem files

-v Run Men2XNF8 in verbose mode

-h Display Men2XNF8 syntax

PLD_DVE

PLD_DVE creates an XNF viewpoint for a Xilinx design. You only have to run PLD_DVE once for each design.

Syntax

pld_dve ***design tech_type*** **[*design_viewpoint_name*]**

Variables

design is the file name of the component declared as root of the design hierarchy.

tech_type specifies the PLD family, either XC2000, XC3000, XC4000, or XC7000.

design_viewpoint_name specifies the name of the design viewpoint to generate. The default name is XNF, which is placed in the design directory.

Options

-h Display syntax for PLD_DVE

PLD_DVE_BA

Note: A default viewpoint must exist before running PLD_DVE_BA.

PLD_DVE_BA reads the Mentor Back Annotation (MBA) file created by XNFBA and back annotates the original default viewpoint created during functional simulation. The back annotated viewpoint is then read by QuickSim II.

Syntax

pld_dve_ba *component_name design*.mba
[design_viewpoint] [-help]

Variables

design.mba is the Mentor Back Annotation file created by XNFBA.

Options

design_viewpoint	**Specify different viewpoint other than default**
-help	**Display syntax for PLD_DVE_BA**

PLD_DVE_SIM

PLD_DVE_SIM reads in your design and creates the default viewpoint that is used by the Mentor Graphics QuickSim II simulator. You only have to run PLD_DVE_SIM once for each design. Once the default viewpoint is created, you can make subsequent edits to the design and then immediately execute QuickSim II without running PLD_DVE_SIM.

Syntax

pld_dve_sim *design tech_type*
[*design_viewpoint_name*] [-help]

Variables

design is the name of the component declared as root of the design hierarchy.

tech_type specifies the PLD family, either XC2000, XC3000, XC4000, or XC7000.

design_viewpoint_name specifies the name of the design viewpoint to generate.

Options

design_viewpoint	**Specify different viewpoint other than default**
-help	**Display syntax for PLD_DVE_SIM.**

QUICKSIM II

The QuickSim program loads your design into the Mentor Graphics QuickSim II simulator.

Syntax

```
quicksim design
```

Variables

design is the name of the component declared as root of the design hierarchy.

Options

See Mentor Graphics QuickSim II documentation.

TIMSIM8

TIMSIM8 prepares a Xilinx routed design for QuickSim II timing simulation. It runs LCA2XNF, VMH2XNF, UNAKAXNF, XNFBA, PLD_DVE_BA, Gen_Sch8, and PLD_DVE_SIM as needed.

Syntax

```
timsim8 design {-o | -g}
     [-q -verbose -help]
```

Variables

design is the name of your Mentor design directory.

Options

-o Use original schematic for simulation

-g Generate schematic for simulation

-q Run QuickSim II

-verbose Run TIMSIM8 in verbose mode

-help Display TIMSIM8 syntax

UNAKAXNF

The UNAKAXNF program restores the net names that were aliased in the *design*.aka file, created by MAP2LCA.

Syntax

```
unakaxnf [-options] xnf_filename[.xnf]
```

Variables

xnf_filename.xnf is a routed XNF file. It is output by LCA2XNF, and is used as input by XNFBA.

Options

-a *aka_file*.aka Specifies the AKA File Name

This option changes the input AKA file name to something different from the default AKA file name. The default file name is *xnf_filename*.aka.

-o *output_file*.xnf Specifies Alternate Output XNF File Name

This option specifies an alternate output XNF file name that is different from the default XNF file name. The default file name is *xnf_filename*.xnf.

-w Overwrite Output XNF

The -w option automatically overwrites the output XNF file without asking if you are certain that you want to overwrite it.

VMH2XNF

VMH2XNF creates an XNF file with timing parameters for use in timing simulation. The input file can be a .vmh or .vmd (for XC7272 designs) file. Refer to the *XEPLD Reference Guide* for more information on syntax, variables, and options.

X-BLOX

This program creates an XGS file and one or more XNF files from the XTF file created by XNFPrep. The XGS file contains a list of all the X-BLOX primitives used in the original schematic and their associated XNF simulation models created by X-BLOX.

Syntax

xblox *design*.xtf simdir=*simdir* *sim*=xnf

Variables

design.xtf is created by XNFPrep.

simdir is the directory used for your functional simulation models

sim - If SIM=XNF, X-BLOX outputs an XGS functional simulation model.

The following figure illustrates the directory structure that is created when X-BLOX processes your design.

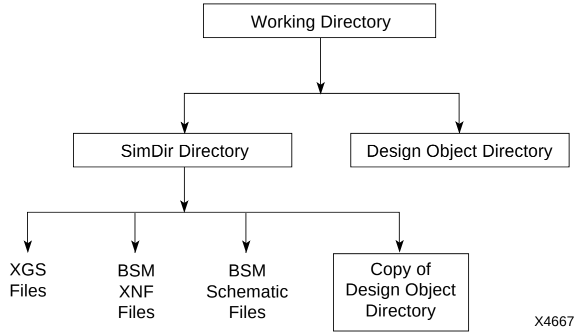


Figure 10-11 X-BLOX Directory Structure

XBLXGS

This program creates a new schematic that looks like the original one. In this new schematic, the X-BLOX symbols have been replaced with symbols that have the same footprint but now have fixed bus pin widths and simulation models. This new schematic can be simulated without having to run PPR.

Syntax

```
xblxgs design xgs_filename[.xgs] [options]
```

Variables

design is the name of the component declared as root of the design hierarchy.

xgs_*filename* is the XGS file created by X-BLOX that describes how the simulation schematic should be drawn.

Options

-w Silent overwrite

-d Output directory name; default is simdir

-o Output design name

XDelay

Note: This program only needs to be run on XC3000A, XC3000L, XC3100A, and XC4000 designs. APR adds net delays to XC2000, XC2000L, XC3000, and XC31000 designs.

The XDelay program adds net delays to a routed LCA file. For more information on XDelay, see the *XACT Reference Guide*.

XNFBA

The XNFBA program combines the pre-route XNF file and the post-route XNF file into a new file that has the original symbol and signal names with post-route delays. This program works with XC2000, XC3000, and XC4000 families.

Note: If you do not have LCA2XNF version 4.30 or higher, you cannot use XNFBA.

Syntax

```
xnfba -a design.xnf -b design_routed.xnf
        [-options]
```

Variables

design.xnf is the pre-routing flat XNF/XG file that is used as input to APR or PPR.

*design*_routed.xnf is the post-routing flat XNF file generated by LCA2XNF with the -g option.

Options

-a *design*.xnf Specify Pre-Routing Flat XNF file

The -a option specifies the pre-routing flat XNF/XG file that is used as input to APR or PPR. If you have hierarchical XNF files, you can use the XNFMERGE program to flatten the design before routing. The -a option can alternatively specify the XG file generated by X-BLOX. The -a option is required and must be consistent with the file for -b option.

-b *design*_routed.xnf Specify Post–Routing Flat XNF file

The -b option specifies the post-routing flat XNF file generated by LCA2XNF with the -g option. The design should be completely routed with no DRC errors, before being processed by LCA2XNF. The -b option is required.

-m Write Mentor Back–Annotation File

The -m option writes a Mentor Graphics Back Annotation File. The file name is the root name of the **-b** XNF file with the extension MBA.

-x Write Back–Annotated XNF file

The -x option writes a back-annotated XNF file. The file name is the root name of the -b XNF file with an XBF extension. The -x option is the default.

Examples

To produce an XC2000, XC3000, or XC4000 back-annotated XBF file, type:

xnfba -a ***test_pre*****.xnf -b** ***test_post*****.xnf -x -m**

where *test_pre*.xnf is the pre-routing flat XNF file and *test_post*.xnf is the post-routing flat XNF file. An output file, *test_post*.xbf, is produced.

To produce an XBF file for an X-BLOX design, type:

xnfba -a ***test_pre*****.xg -b** ***test_post*****.xnf -x -m**

where *test_pre*.xg is the pre–routing flat XNF file and *test_post*.xnf is the post–routing flat XNF file. An output file, *test_post*.xbf, is produced.

Keep all design files consistent. If you change anything in the schematic, repeat the entire design flow, starting with Men2XNF8, before using XNFBA. If you use XDE on the design routed by APR or PPR, XNFBA might not work properly if you make edits or changes.

If you use the MakeBits program with the -t option to save tied nets in your LCA file, you must use the -t option with the LCA2XNF program to remove tied nets from the XNF file.

Verify that the BID file is in the same directory as the LCA file before you run the LCA2XNF program. The BID file must be generated by the same run of PPR that generates the LCA file.

You might notice differences in simulation between the original design that was back annotated using XNFBA with the -m option, and a new schematic generated by Gen_Sch8 directly from the XNF file for the design. The differences in simulation results originate in PPR. One method PPR uses to place and route a design is called logic replication. A block of logic (usually with a large fanout) is duplicated, which gives the PPR program two choices when routing the load pins of the output of the logic block.

For example, start with a single block of logic whose output has 10 load pins, as shown in the figure below.

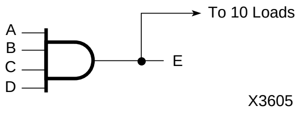


Figure 10-12 Logic Block with 10 Load Pins

PPR can take this block and replicate it, which produces two identical blocks of logic, each with 5 load pins on the output, as shown in the figure below (the numbers next to the input pins are delay values).

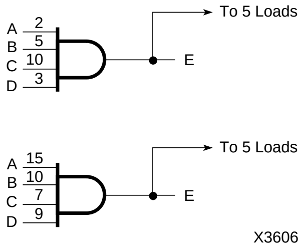


Figure 10-13 Replicated Logic Blocks

The identical blocks ease routing problems and usually the overall delay for the output net as a whole.

Although there is only one path to inputs a, b, c, and d, in the original schematic, there are two delay paths in the back-annotated XNF file for each of them. XNFBA must now use one of the two values to back annotate both paths, even though the values are different. The longest delay values are always chosen, so in this example, a, b, c, and d are assigned delay values of 15, 10, 10, and 9, respectively.

This limitation in simulation using the original schematic probably impacts less than one percent of designs, and only in very small parts. This information is provided so you ensure that your design works in simulation almost as it does in the actual device. Weigh the advantages of being able to use your original schematic against the possibility of a delay value being represented as greater than it actually is. To solve this problem, simulate the back-annotated design as much as possible, to ease problems in finding nets and instances, to compare the simulation output to the schematic, then run simulation using the output of LCA2XNF directly, instead of XNFBA.

XNFMerge

XNFMerge converts a hierarchical design to a “flattened” design that contains no references to other XNF files. Refer to the *XACT Reference Guide* for detailed information on XNFMerge.

XNFPrep

XNFPrep performs a design rule check and removes unused and redundant logic from a flattened XNF file. It also checks the syntax of the XACT-Performance parameters found in the design and prepares delay information for PPR path analysis. Refer to *XACT Reference Guide* for detailed information on XNFPrep.

Mentor Graphics Interface/ Tutorial Guide

Design Architect Tutorial

Chapter 11

Design Architect Tutorial

This chapter steps you through a basic FPGA design procedure from schematic entry using Design Architect to verification of the design on a demonstration board. Information on using the Mentor Graphics Design Manager graphic user interface configured for Xilinx designs is also included. The simple design example used in this tutorial demonstrates many system features that you can apply to more complex FPGA and EPLD designs.

Note: Although this tutorial describes creating and processing FPGA designs, you can apply most of the steps to EPLD designs. See the "XEPLD Tutorial" chapter for EPLD-specific information.

This tutorial includes the following:

- Installing the tutorial files
- Using Mentor Graphics Design Manager
- Targeting the tutorial design (Calc) for an XC4000 device
- Using Design Architect
- Completing the ALU block in the Calc design
- Exploring Xilinx library elements
- Exploring XC3000/XC4000 oscillator
- Inverting output display signals
- Controlling FPGA layout from the schematic
- Editing the Calc design for an XC4000 device
- Configuring XMake using Xilinx Design Manager (XDM)
- Converting design to XNF file using PLD_Men2XNF8

- Implementing design using PLD_XMake
- Examining routed designs with Xilinx Design Editor (XDE)
- Verifying Calc design on a demonstration board
- Making incremental design changes
- Command Summaries

Required Background Knowledge

This tutorial assumes that you have a basic understanding of the following:

- UNIX Operating System. None of the command sequences are given in AEGIS.
- Motif Windows. Mentor Graphics applications conform to the Motif window style.

Note: When you are instructed to close a window, it is important that you exit from the window rather than iconize it.

- Some knowledge of Design Manager, Design Architect, and Xilinx core software. For more information on these applications, refer to the list of related publications at the beginning of this user guide.

FPGA Design Flow

A general overview of the FPGA design flow is shown in the following figure.

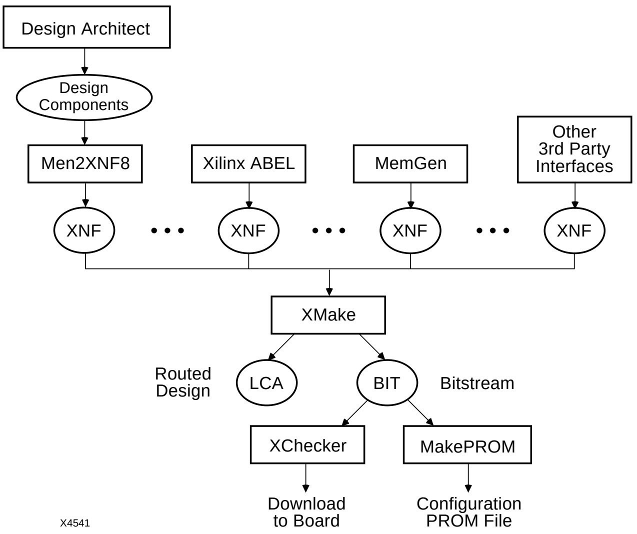


Figure 11-1 FPGA Design Flow

This process can be executed automatically by using the PLD_DA, PLD_Men2XNF8, and PLD_XMake icons found in the Xilinx-configured Design Manager (PLD_DMGR).

The steps in the FPGA design flow are as follows:

1. Create a schematic using Design Architect with symbols from the Xilinx libraries.
2. Use PLD_Men2XNF8 to translate the Mentor Graphics design file to a Xilinx netlist (XNF) file. PLD_Men2XNF8 performs the following:
   - Creates a viewpoint that specifies how to generate a .EDIF file
   - Uses the Mentor Graphics EDIF netlister ENWRITE, to create the .EDIF file from the design
   - Converts the .EDIF file to hierarchical Xilinx Netlist Format (XNF) file
3. Use PLD_XMake to read the netlist produced by PLD_Men2XNF8. PLD_XMake performs the following:

   - Merges the hierarchical XNF files into a single top-level XNF file
   - Trims unused logic
   - Performs an electrical rule check on the XNF file
   - Partitions the logic into LCA device resources, Configurable Logic Blocks (CLBs), and input/output blocks (IOBs)
   - Places the blocks and routes the connections between them
   - Creates a configuration bitstream

4. Download the bitstream into one of the Xilinx demonstration boards to test the design

An incremental design methodology is described in this tutorial. In incremental design, the design is processed; a small change is made to the design; and then the design is processed again. Place and route information from the previous design processing cycle is used to constrain subsequent cycles of the same design. When this method is used, timing information in a design remains relatively stable through many processing cycles. Also, place and route time is considerably reduced since much of the processing is done in previous cycles.

The tutorial design can be targeted for an XC3000A, XC3000, XC4000A, or XC4000 device. You can use a Xilinx demonstration board to test the functionality of your design. Make sure your demonstration board and software support your selected device. To determine compatibility, refer to the release notes that came with your software package.

In this tutorial the following conventions are used to refer to the various device families:

- XC3000 family: includes XC3000, XC3000A, XC3000L, XC3100, and XC3100A devices
- XC4000 family: includes XC4000, XC4000A, and XC4000H devices
- XC2000 family: includes XC2000 and XC2000L devices

Software Installation

Required Software

The following versions of software are required to perform this tutorial:

- Mentor Graphics Version 8.2_5 or later
- Xilinx / Mentor Graphics Interface DS344 Version 5 or later
- XACT Design Manager (XDM) Version 5 or later

Before Beginning the Tutorial

Before beginning the tutorial, set up your workstation to use Mentor Graphics and XACT Development System software as follows:

1. Verify that your system is properly configured. Consult the release notes that came with your software package for more information.
2. Install the following sets of software:
   - XACT Development System (DS501 or DS502) Version 5.00
   - Xilinx DS344 Mentor Graphics Version 5 interface
   - Mentor Graphics software Version 8.2_5 or later, including Design Manager, Design Architect, QuickSim II, QuickPath, as well as the software needed to produce EDIF netlists from ENWRITE, which requires special licensing
3. Verify the installation. When you finish the installation, verify that your .cshrc or setup file contains lines similar to the following:

Note: Path names of directories will vary. For more information on paths and environment variables, refer to the release notes that came with your software package.

```
setenv LCA /location_of_ds_344:
setenv XACT /location_of_ds_344:
/location_of_ds502
set PATH=($PATH                    \
   $LCA/com/sparc                  \
   $LCA/bin/sparc                  \
  /location_of_ds502 /bin/sparc    \
   )
```

Modifying Mentor Graphics Variables

Make sure that the following Mentor Graphics specific variables are set correctly:

- MGC_HOME

  This should point to the Mentor Graphics software tree.

- MGC_GENLIB

  This should point to the Mentor Graphics gen_lib library, normally $MGC_HOME/gen_lib.

- LD_LIBRARY_PATH

  This variable is used by the Mentor Graphics Design DataPort (DDP) routines that are accessed by some Xilinx programs. On a SPARC station with OpenWindows installed in /usr, this variable is set as follows:

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

- MGC_LOCATION_MAP

  This variable should point to a valid location map file.

  Every symbol and schematic in your design contains a reference. A reference indicates where the design object resides on your disk. The tutorial designs use variables in their reference definitions so they can be easily relocated. All of the tutorial designs use the variable, $xilinx_tutorial, to define the path reference. $xilinx_tutorial must be defined in the file pointed to by $MGC_LOCATION_MAP. For example, the design object, led_inv in the *install_path*/tutorial/mentor/calc_da directory, uses the path reference $xilinx_tutorial/calc_da/led_inv to define where it is located in the directory structure. If the tutorial directories were copied to the path, /home/bclinton/mentor/xtutorial, the following two lines must be added to the file pointed to by $MGC_LOCATION_MAP:

$xilinx_tutorial
/home/bclinton/mentor/xtutorial

If a query was made to determine where the design object '$xilinx_tutorial/stack' is located, the Mentor Graphics tools would use this definition to determine that stack is at /home/bclinton/mentor/xtutorial/calc_da/stack.

It is also important that the $LCA variable be instantiated, but not defined, in the file pointed to by $MGC_LOCATION_MAP. To do this, add the following line to MGC_LOCATION_MAP, followed by an empty line:

LCA
(empty line)

Refer to the Mentor Graphics documentation for more information on location maps.

- MGC_WD

  This variable should point to the working directory. For the tutorial, it should point to the directory where the tutorial is worked on.

- LCA

  In addition to instantiating it in the file pointed to by MGC_LOCATION_MAP, the LCA environment variable should point to the directory where the DS344 software is installed.

Installing the Tutorial

The tutorial files are optionally installed when you install the DS344 interface software. If you have already installed the software, but are not sure whether you specified tutorial installation, check for a tutorial directory under your DS344 directory. The tutorial directory contains the tutorial files.

Standard Directory Structure

When a design object is created in Mentor Graphics, a directory is created in the project directory with the same name as the design object. This directory contains a schematic directory, symbol files, viewpoint files, and part interfaces. The directory is identified as a design object by the file, *design_nam*e.mgc_component.attr, that resides at the same level as the directory which has the name. For example, if a schematic named calc is created, a calc directory is created, and at the same level the file, calc.mgc_component.attr, is created. The calc directory will contain all the files that describe calc.

Note: In this tutorial, file names and directory names are in lower case and the design example is referred to as Calc.

Tutorial Directory and Files

You will complete the Calc design in this tutorial. During the tutorial installation, the $LCA / tutorial directory is created; design object directories are created; and the tutorial files needed to complete the design are copied to the calc_da directory. Some of the files you need to complete the tutorial design are not copied, because you will create these files in the tutorial. However, solutions directories with all input and output files are provided. They are located in the $LCA / tutorial / mentor directory and are listed in the following table:

Table 11-1 Tutorial Design Directories

Directory	Description
calc_da	Tutorial Directory
calc_3k	Solution Directory for XC3020PC68
calc_3ka	Solution Directory for XC3020APC68
calc_4k	Solution Directory for XC4003PC84 and XC4003APC84

The solutions directories contain the design files for the completed tutorial, including schematics and the bitstream file. To conserve disk space, some intermediate files are not provided, except in the calc_3k directory, which is complete. Different intermediate files are created for different device families. Do not overwrite any files in the solutions directories.

The calc_da directory contains the incomplete copy of the tutorial design. The installation program copies a few intermediate files to the calc_da tutorial directory, and you will create the remaining files when you perform the tutorial. As described in a later step, you will copy the calc_da directory to another area and will perform the tutorial in this new area. The following table lists and describes the directories and files in the calc_3k solution directory.

Table 11-2 Tutorial Directories/Files in the calc_3k Directory

Directory or File Name	Description
calc	Top-level design directory
control	Design directory for control module
statmach	Design directory for state controller module
alu	Design directory for ALU module
alu_blox	X-BLOX version of ALU design component (see “X-BLOX Tutorial”)
bloxsoln	Design component for ALU module, X-BLOX version (see “X-BLOX Tutorial”)
muxblk2	Design component for arithmetic function in ALU
andblk2	Design component for arithmetic function in ALU
orblk2	Design component for arithmetic function in ALU
xorblk2	Design component for arithmetic function in ALU
muxblk5	Design component for multiplexer of arithmetic outputs in ALU
stack	Design component for stack
7seg_dec	Design component for seven segment decoder
debounce	Design component for debounce circuit

Directory or File Name	Description
osc_3k	Design component interface to RC circuit on demonstration boards; generates clock
*.xnf	Xilinx netlist format files created by Men2XNF8
edif2xnf.log	EDIF2XNF log file
men2xnf8.log	Men2XNF8 log file
xnfprep.log	XNFPrep log file
calc.bit	Bitstream for downloading to LCA; generated by MakeBits
calc.crf	Cross-reference file generated by XNFMap
calc.edif	EDIF netlist produced by ENWRITE
calc.lca	Placed and routed design file; generated by PPR
calc.lcb	Placed and routed design file, unoptimized.
calc.mak	Script file generated and used by XMake
calc.map	Partitioned logic file generated by XNFMap
calc.mbo	Bitstream configuration file generated by MakeBits
calc.mrg	Merge report file generated by XNFMerge
calc.odf	Intermediate version of LCA file generated by PPR
calc.out	XMake report file
calc.pgf	Partitioning guide file generated by XNFMap, needed for incremental design process
calc.prp	Report file generated by XNFPrep
calc.rpf	PPR report file for unoptimized .lcb file
calc.rpt	Routing report file generated by PPR
calc.xff	Output of XNFMerge, netlist of merged design
calc.xtf	Output of XNFPrep, netlist of trimmed design
calc_3k.cst	Constraints file that sets pad locations for 3k parts
calc_3k.do	QuickSim II command file for 3k, 3ka simulation
stat_abl.abl	Xilinx ABEL file for state controller module, replaces Statmach schematic (see “Xilinx ABEL Tutorial”)

Note: In addition to the files listed above, there is a *filename*.mgc_component.attr file associated with each design component directory. This file identifies the corresponding directory as a Mentor Graphics design component.

Starting the Design Manager

To start the Design Manager configured for Xilinx designs, type the following at the operating system command line:

```
> pld_dmgr ↵
```

The following figure appears:

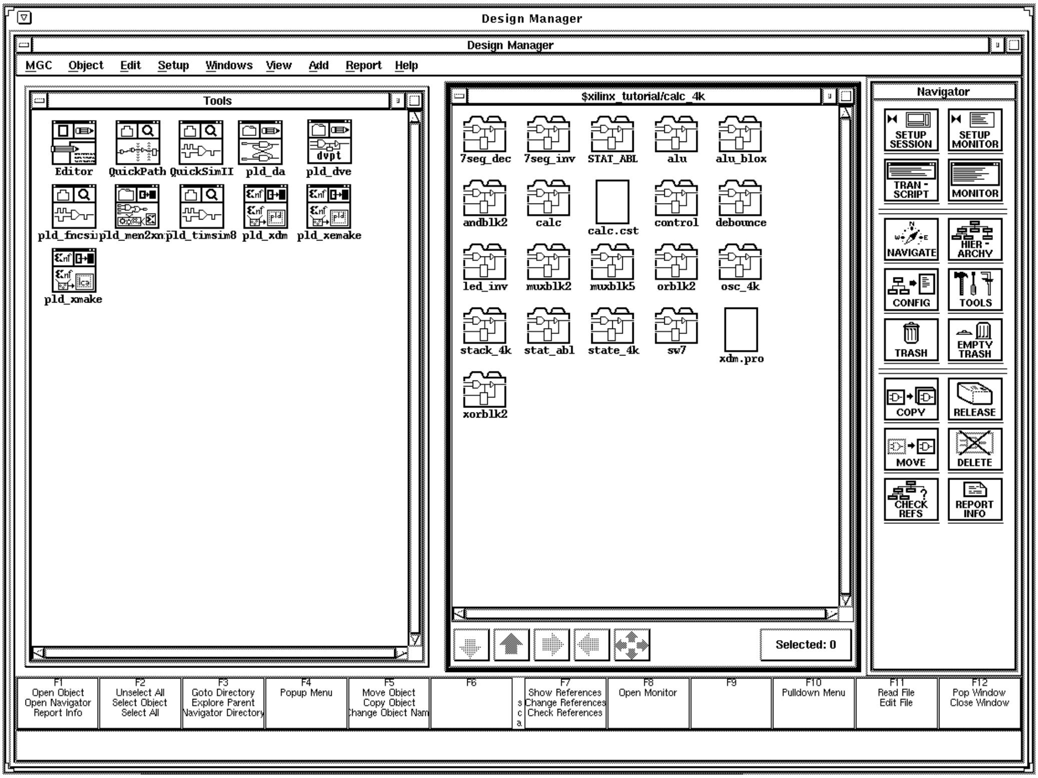


Figure 11-2 Design Manager Window

There are three sub-windows in the Design Manager window: the Tools Window, the Navigator Window, and the Command Palette. Each sub-window is described below.

Mentor Graphics windows conform to Motif standards. You should know how to move, close, and minimize (or iconize) the Motif windows. When multiple windows are open, the active window has a blue border and inactive windows have a grey-brown border. For more information on Design Manager operation, refer to the Mentor Graphics documentation.

Tools Window

The Tools Window on the left contains icons representing all the Mentor Graphics and Xilinx applications you need to execute the steps in the Xilinx design flow.

The Tools window contains the following Xilinx-specific icons:

- PLD_DA: Design Architect configured for Xilinx designs.
- PLD_DVE: Creates design viewpoints necessary for the creation of .EDIF files from a design.
- PLD_Men2XNF8: Creates a Xilinx Netlist Format (XNF) file from your design. You must run PLD_Men2XNF8 on most designs before running PLD_FNCSIM8. You must run this program at least once before you run PLD_XMake. Only run PLD_Men2XNF8 on design objects, and not on any other type of file.

Note: Any time a schematic change is made, PLD_Men2XNF8 must be run again before PLD_XMake or PLD_FNCSIM8 is run, so that the schematic change is reflected in the netlist.

- PLD_FNCSIM8: Performs all the steps necessary to prepare a design for functional simulation.
- PLD_XMake: Reads the XNF file created by PLD_Men2XNF8, creates an LCA file, and creates a BIT file. The BIT file is used to configure a Xilinx FPGA. Only run PLD_XMake on XNF files, which are represented in the navigator window as a file icon with the word “XNF” on it.
- PLD_XEMake: Reads the XNF file created by PLD_Men2XNF8, automatically fits your design into a selected EPLD device, and generates a device program file.

- PLD_TIMSIM8: Performs all the steps necessary to prepare a design for timing simulation. Run after PLD_XMake.
- PLD_XDM: Starts the Xilinx Design Manager (XDM).

Navigator Window

Use the Navigator window to move around the directory hierarchy and select files, folders, and other types of design objects.

The Navigator has three buttons located at the bottom of the window. The two buttons on the left have up and down arrows on them. Use these buttons to move up and down the directory hierarchy. To move down the hierarchy with the down arrow, you must first select the desired folder in the Navigator. The right-most button has four arrows on it, one pointing in each direction. When you select this button, a dialog box appears and you can type in the path to the directory you want to display in the navigator window. Using this button is sometimes quicker and easier than using the up and down arrows.

Command Palette

Use the Command Palette to access the most commonly used Design Manager menu items.

Copying the Tutorial Files

Perform the following steps to make a working copy of the tutorial files:

1. In the navigator window, move to the directory where the tutorial files were installed.
2. Select the calc_da directory.
3. Choose **Right Mouse Button → Edit → Copy:** A dialog box appears.
4. In the dialog box, type the directory path where you want the working copy of the tutorial files copied. For example, if you want to copy the files to /home/dum/tutor/mentor, enter /home/dum/tutor/mentor/calc_da.

5. Use the navigator to change directories to the location of the working copy of calc_da.
6. If necessary, modify MGC_LOCATION_MAP so that the $xilinx_tutorial variable points to the directory where the copy of calc_da is located.

Targeting the Design for the XC4000 Family

The incomplete calc_da design is configured for a XC3020APC68 part or a XC3020PC68 part. If you want to target a demonstration board with one of these devices, go to the next section "Starting Design Architect." If you are targeting the tutorial design for a 4003APC84 or a XC4003PC84 device, you must convert the design to reference the XC4000 library instead of the XC3000 library.

The procedure provided below allows you to change every reference of every design object in the $xilinx_tutorial/calc_da directory from the XC3000 library to the XC4000 library. Since the designs were created using the Unified Libraries, the parts in the XC3000 and XC4000 libraries have identical footprints and pinouts. This allows you to easily retarget designs to a different device family, provided only library parts common to the two families are used. You must manually replace any library parts that are not common to both families. For example, if a gclk is used in an XC3000 design that is retargeted for use in an XC4000 device, you must manually replace the gclk with a bufgp or bufgs, which are the XC4000 equivalents of a gclk.

To change the references, perform the following steps:

1. Select **MGC → Location Map → Set Working Directory** from the menu bar. A small dialog box appears at the bottom of the screen.
2. Type **$xilinx_tutorial** in the Directory field of the dialog box, then select **OK** or press return. This sets the working directory to the directory above the calc_da directory. This allows you to make changes to the references of all the files in the calc directory.
3. In the navigator window, select the $xilinx_tutorial/calc_da directory using the left mouse button.
4. Select **Right Mouse Button → Edit → Change → References.**

5. A dialog box appears. In the “from” box, type **xc3000**; in the “to” box, type **xc4000**. Press return or select **OK**.
6. Whenever you change references in a design, you should also check and save it in Design Architect, to verify that the reference changes were successful. This will be done as part of the tutorial.

Starting Design Architect

To open the Calc design in Design Architect, perform the following steps:

1. Select **MGC → Location Map → Set Working Directory** from the menu bar. A small dialog box appears at the bottom of the screen.
2. Type **$xilinx_tutorial/calc_da** in the Directory field of the dialog box, then select **OK** or press return. This sets the working directory to the directory where you will work on the tutorial.
3. Select the $xilinx_tutorial / calc_da / calc design object in the navigator window.
4. Select **Right Mouse Button → Open → PLD_DA**. The Design Architect window appears and displays the Calc design as shown in the figure below.
5. Resize the Design Architect window to cover the entire screen.

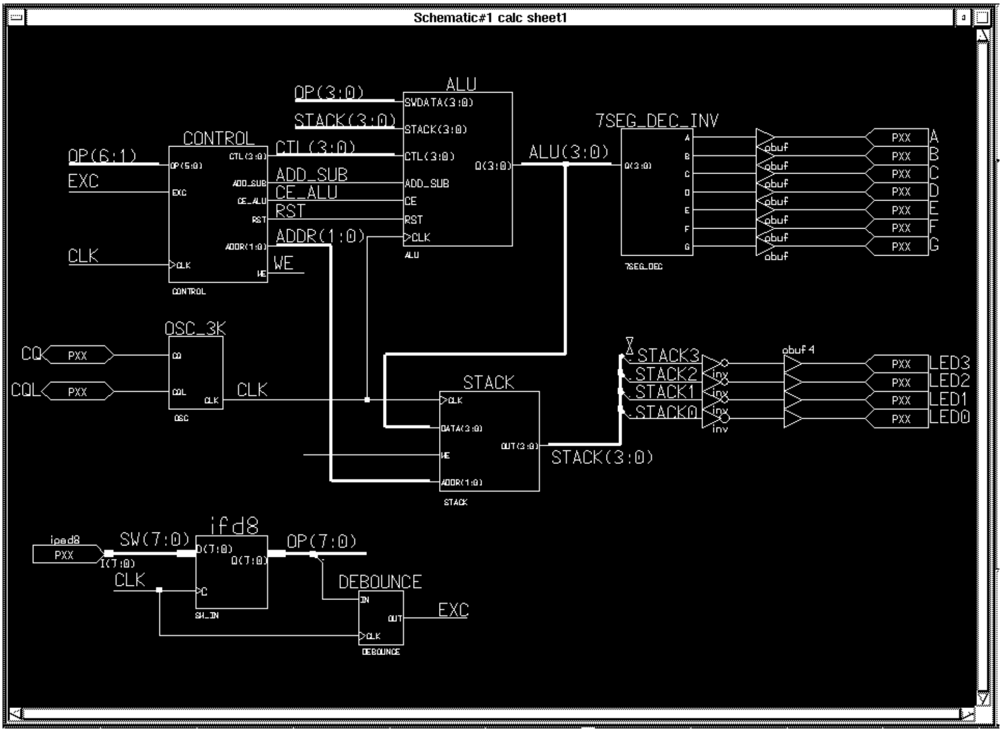


Figure 11-3 Top-Level Schematic for Calc

Using the Mouse in Design Architect

Left Mouse Button

Use this button to select or de-select objects on a sheet. A selected object has a white dashed outline. Hold down this button and drag the mouse to select multiple objects.

Middle Mouse Button (Strokes)

Use the middle mouse button to perform actions known as strokes. You can use strokes as shortcuts to perform common tasks. Perform a stroke by pressing and holding the middle mouse button while moving the mouse to draw a line with a specific shape. The shape you draw is converted to a number string by Design Architect to determine which command is executed. The number is determined based on the figure below:

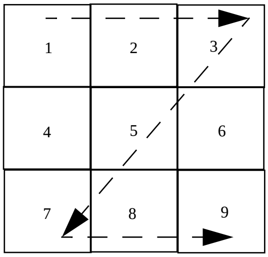


Figure 11-4 Using Strokes, Example of "Z" stroke (1235789)

For example, a "Z" stroke represents the number 1235789. To determine the commands that the strokes represent, select **`Help`** → **`On Strokes`** from the menu bar at the top of the screen. You can also hold down the middle mouse button and drawing the shape of a question mark ("?") to display the stroke help screen. When applicable, strokes are used in this tutorial.

Right Mouse Button

Use this button to display different menus depending on the object(s) selected on the schematic sheet. For example, if a net is selected when the right mouse button is pressed, the net menu appears. Other menus can be accessed regardless of what is selected by using the "Other Menus" selection that appears at the top of each of the menus.

Using the Function Keys

You can also use the keyboard function keys to execute various commands. The boxes at the bottom of the Design Architect window reference the function keys. Each box contains three commands.The top command is executed by pressing the associated function key; the middle command is executed by pressing the function key while holding down the shift key; and the bottom command is executed by pressing the function key while holding down the control key.

Selecting Commands from the Menu Bar

Use the left mouse button to select commands from the menu bar at the top of the screen.

Selecting Commands from the Palette

Use the left mouse button to select commands from the command palette at the right side of the screen. The set of red buttons at the top of the palette change the commands that are available in the palette. The commands displayed in the palette vary depending on what type of window is active in Design Architect. For example, if a symbol editor window is active, commands such as Add Pin, Draw Rectangle, and other commands associated with creating symbols are available in the palette. If there are no windows open in Design Architect, commands such as OPEN SHEET or OPEN SYMBOL are available.

You may need to scroll the palette to access some of the commands by moving the cursor into the palette and using the PageUp and PageDown keys. You can also select **`Right Mouse Button` → `Show Scroll Bars`** to display scroll bars.

Entering Commands from the Keyboard

You can type commands anywhere in the Design Architect window. A dialog box appears at the cursor location to capture the command text. For example, a schematic sheet can be opened by typing the command “open sheet” in the Design Architect window.

Cancelling Commands

When you select a command, it is displayed in either a small rectangular box in the lower-left area of the screen, or in a larger dialog box. In either case, you can cancel commands by selecting the cancel button in the box or by pressing the escape key.

Repeating Menu Commands

You can repeat commands that were executed by either using the menu bar or the menus accessed through the right mouse button by holding down the control key, moving the cursor to the appropriate area, and pressing the right mouse button. For example, if **`Right Mouse Button`** → **`Properties`** → **`Add`** was the last command sequence performed, you can repeat this sequence by holding down the control key and pressing the right mouse button with the cursor in the window where the command was last executed. To repeat the command **`File`** → **`Save`** from the menu bar, move the mouse to the File selection in the menu bar, hold down the control key, and press the right mouse button.

Manipulating the Screen

To zoom in on a specific area of the screen, hold down the F8 key and move the mouse to create a box around the area you want to zoom on.To view the entire schematic, hold down the shift key and press F8 (you can also perform the commands with the strokes 159 and 951, respectively). The schematic can also be zoomed in or out with the menu bar commands **`View`** → **`Zoom In`** and **`View`** → **`Zoom Out`** (or the strokes 357 and 753).

Completing the Calc Design

To complete the tutorial design, you need to add a few design objects to the Calc schematic using Design Architect.

If you need to stop the tutorial at any time, be sure to save the work you have done by first selecting **`Check`** → **`Sheet`** from the menu bar. A window appears containing the results of the design rule check. After reviewing the contents of this window, close it and reselect the schematic window. Then select **`File`** → **`Save`** from the menu bar to save the design. It is important to check your design first before saving it.

Design Description

The top-level schematic of the Calc tutorial design has been created for you. Each of the blocks in the schematic, such as CONTROL or ALU, is linked to a second-level module that describes its logic. Additionally, any second-level module can contain another block that references a third-level drawing, and so on. This organization is known as a hierarchical structure.

In this tutorial, you add three symbols to the ALU block schematic to complete it. First, you create the ANDBLK2 and ORBLK2 symbols and their underlying schematics and then add them to the schematic. Additionally, you copy the FD4CE symbol from the Unified Libraries to the ALU block.

The Calc design is a four-bit processor with a stack. The processor performs functions between an internal register and either the top of the stack or data input from external switches. The results of the various operations are stored in the register and displayed in hexadecimal on a seven-segment display. The top value in the stack is displayed in binary on a bar LED.

The design consists of the following six functional blocks:

- ALU

  The arithmetic functions of the processor are performed in this block.

- CONTROL

  The opcodes are decoded into control lines for the stack and ALU in this module.

- STACK

  The stack is a four-nibble storage device. It is implemented using flip-flops in the device-independent design. You can substitute the RAM module, STACK_4K, in the XC4000 design to take advantage of the on-chip RAM capability of the XC4000 family.

- OSC_3K

  This module is used in XC3000 family designs. It generates a clock signal using the RC oscillator circuit on the XC3000 / XC4000 and XC3000 demonstration boards. It is replaced by the OSC_4K internal oscillator circuit for the XC4000 design.

- DEBOUNCE

  This circuit debounces the “execute” switch, providing a one-shot output.

- 7SEG_DEC

  This block decodes the output of the ALU for display on the seven-segment decoder. This block can be replaced by the

7SEG_DEC_INV component for use on demonstration boards with inverted sense on their 7 segment display.

- IFD8

  The IFD8 is a macro from the Xilinx libraries. It consists of eight input flip-flops, which are used to latch the switch data.

Before proceeding, close (quit) the Calc schematic window. If a dialog box appears asking if you want to save any changes, choose **NO**.

Creating the ANDBLK2 Symbol

Opening a Symbol Window

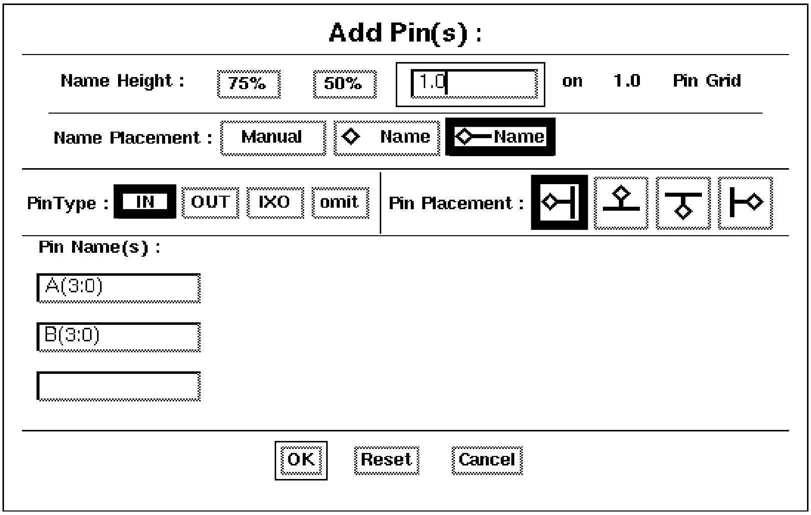
1. Use the left mouse button to select **Open Symbol** in the command palette.
2. Type **$xilinx_tutorial/calc_da/andblk2** in the Component Name box, then select **OK**. A symbol editor window appears.

Creating the Symbol Outline

1. Zoom in until the grid space markers, represented by small crosses, are visible in the symbol window.
2. Select **ADD RECTANGLE** from the palette.
3. Position the cursor in the upper left corner of the symbol window and press the left mouse button.
4. While holding down the left mouse button, move the cursor diagonally to the opposite corner of the symbol window to draw a rectangle that is six grid squares high by eight grid squares wide. Be sure to measure using the grid marks, and not the small dots that define fractions of grid spacing.

Adding Pins to the ANDBLK2 Symbol

1. Select **Add Pin** from the palette. The dialog box in the following figure appears.
2. Fill in the Dialog box exactly as shown and then select **OK.**



Add Pin(s) :

Name Height : 75% 50% 1.0 on 1.0 Pin Grid

Name Placement : Manual ◇ Name [✓ ◇—Name]

PinType : [✓ IN] OUT IXO omit

Pin Placement : [✓ ◇⊣] ⊥ ⊤ ⊢◇

Pin Name(s) :

A(3:0)

B(3:0)

Figure 11-5 Add Pin(s) Dialog Box

3. A small crosshair appears under the cursor, and a rectangular box appears stating that the first pin, A(3:0), is to be placed. Place it as shown in the figure below by moving the cursor to the position where the diamond appears in the figure (one grid space to the left of the rectangle) and pressing the left mouse button. Small purple diamonds indicate pins.

   If you make a mistake before placing a pin, press the escape key to cancel the command, then repeat the above steps. If you make a mistake after placing a pin, press the F2 key to unselect everything. Select the pin (diamond) and the line next to it and press and hold CTRL-F2 to execute a move command. Move the pin to the correct position and release the keys.

4. Place pin B(3:0).

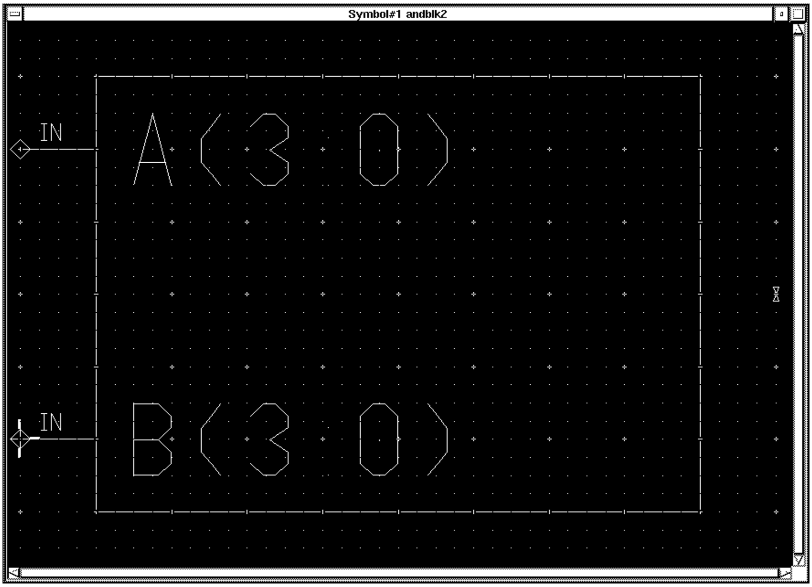
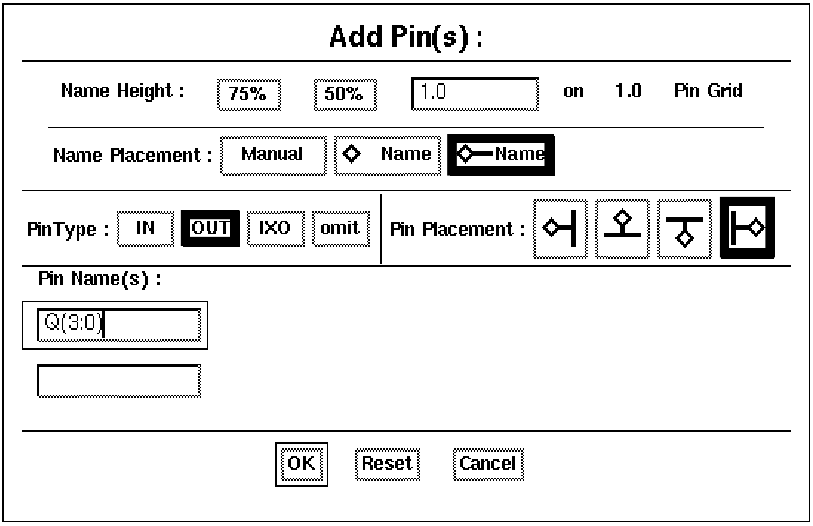


Figure 11-6 Adding Pins A(3:0) and B(3:0)

5. Select **`Add Pin`** from the palette and fill in the dialog box as shown below, then select **`OK`**. Be sure to set the name height to 1.0.



Add Pin(s) :

Name Height : 75% 50% 1.0 on 1.0 Pin Grid

Name Placement : Manual ◇ Name [✓ ◇—Name]

PinType : IN [✓ OUT] IXO omit

Pin Placement :

Pin Name(s) :

Q(3:0)

Figure 11-7 Add Pin(s) Dialog Box

6. Place the pin Q(3:0) as shown in the figure below.

7. To adjust the positioning of the pin names, move the mouse over the text, press and hold the F7 function, and move the mouse to reposition the text. Release the F7 key to place the text at the new location.

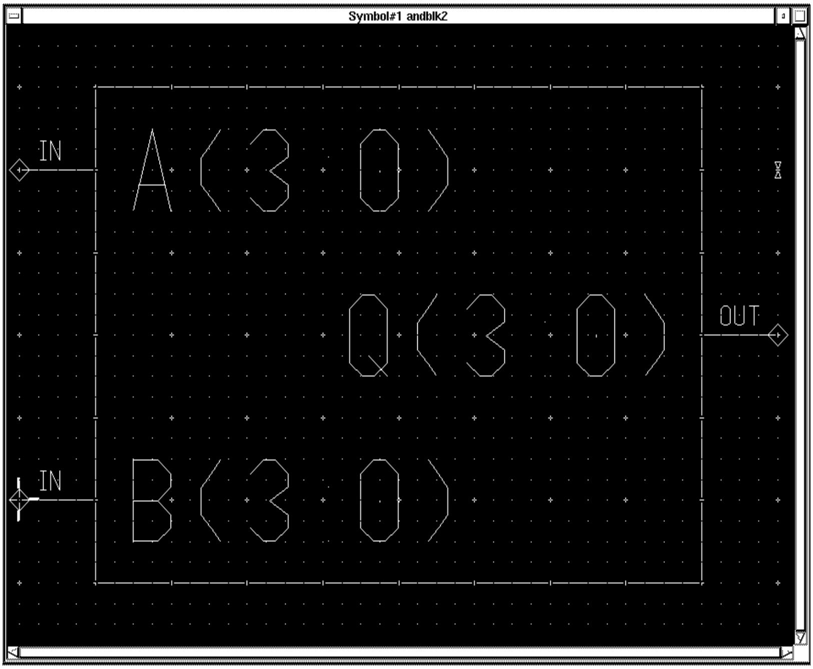


Figure 11-8 Adding Pin Q(3:0)

Adding Text

You can add comment text to a symbol to make it more easily identifiable on a schematic, or to annotate it without modifying its function. To add text to the symbol, perform the following steps:

1. Select the red **TEXT** button at the top of the palette to display the text editing icons.
2. Choose **ADD TEXT** from the palette. A small rectangular dialog box appears in the lower left portion of the window.
3. Type **ANDBLK2** in the Text field of the dialog box, then press return or select **OK**.
4. Move the cursor into the symbol editor window and place the text directly above the symbol body by moving the mouse to the proper position and pressing the left mouse button.

If you make a mistake while typing the text and the text has already been placed, move the mouse over the text and press the F7 key while holding down the shift key. A small dialog box appears at the bottom of the screen containing the selected text. Modify the text in the dialog box. Select **OK** to change the text on the symbol. You can use this method to modify any text on the symbol, such as pin names.

Modifying Text Size

To modify symbol text size, perform the following steps:

1. Press the F2 key to unselect everything.
2. Use the left mouse button to select the text, ANDBLK2, at the top of the symbol.
3. Select **Right Mouse Button → Change Height → 1.5 X pin spacing**.
4. Place the cursor over the text and press and hold the F7 key.
5. While still holding down the F7 key, move the text so that it is centered above the symbol body, as shown in the following figure.

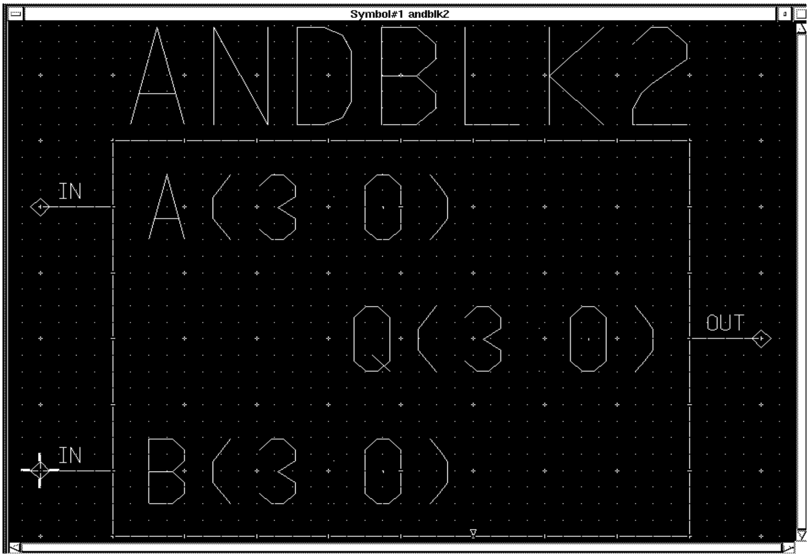
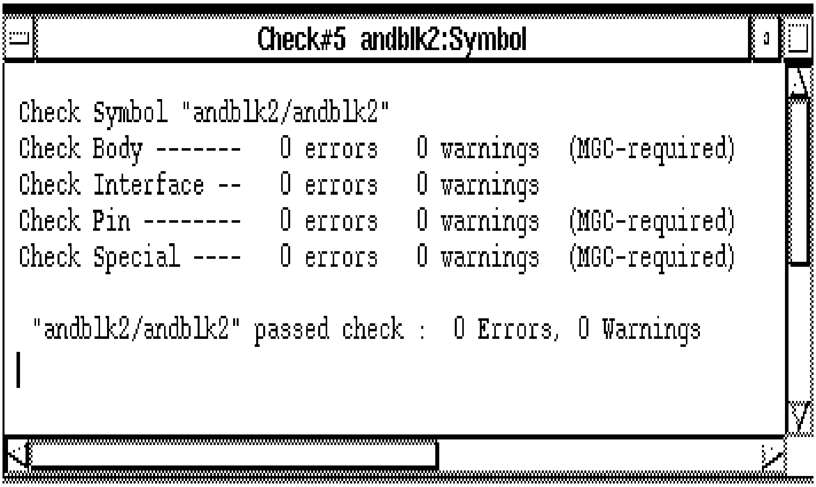


Figure 11-9 Completed ANDBLK2 Symbol

Saving the ANDBLK2 Symbol

To save the ANDBLK2 symbol, perform the following:

1. From the menu bar, select **`Check → With Defaults`**. A text window appears containing the results of the design rule check.
2. Check to see that the information displayed is the same as that in the following figure. If you do not have the same output, correct the symbol to eliminate the differences and then check the symbol again.
3. Close the text window by selecting **`Close`** from the menu that appears when the left mouse button is pressed in the box in the upper left hand corner of the text window.
4. Select **`File → Save Symbol`** from the menu bar to save the symbol.



```
Check Symbol "andblk2/andblk2"
Check Body -------   0 errors   0 warnings  (MGC-required)
Check Interface --   0 errors   0 warnings
Check Pin --------   0 errors   0 warnings  (MGC-required)
Check Special ----   0 errors   0 warnings  (MGC-required)

 "andblk2/andblk2" passed check :  0 Errors, 0 Warnings
```

Figure 11-10 Output from Check

Creating the ORBLK2 Symbol

The next step is to create the symbol for ORBLK2, as shown in the following figure. Since ORBLK2 is similar to ANDBLK2, use the ANDBLK2 symbol and modify the text.

1. Move the cursor above the ANDBLK2 text. Press the F7 key while holding down the shift key to select the **Change Text Value** command.
2. In the small dialog box that appears in the lower left corner, type **ORBLK2** in the New Text field, then select **OK**.

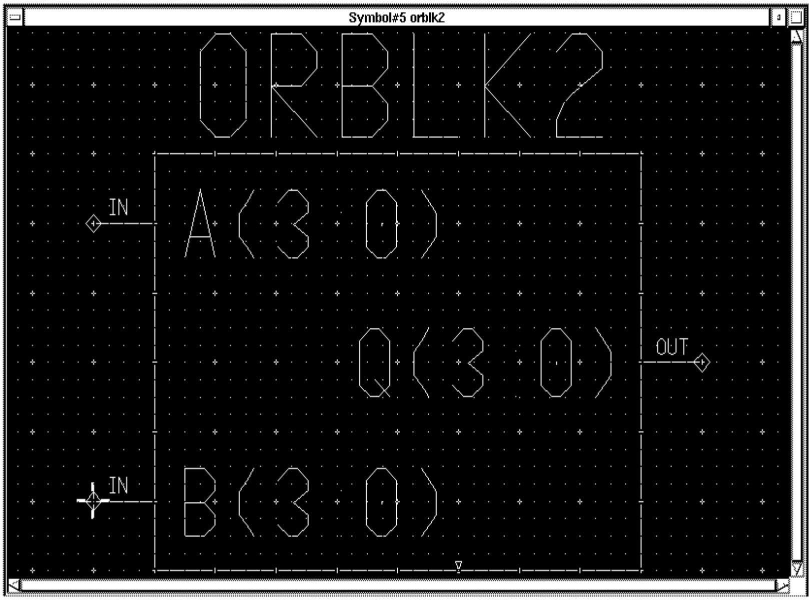


Figure 11-11 Completed ORBLK2 Symbol

3. If necessary, use the cursor and F7 key to move and center the text, as described earlier.

4. From the menu bar, select **`Check → With Defaults`**. A text window appears containing the results of the design rule check. Since you are modifying the ANDBLK2 symbol, the text still refers to ANDBLK2.

5. If any errors are reported in the Check text window, correct them on the symbol and check the schematic again. Otherwise, close the text window.

6. To save the symbol as ORBLK2, select **`File → Save Symbol AS...`**. A dialog box appears.

Warning: It is important that you select the Save Symbol As command instead of Save to prevent overwriting the original ANDBLK2 file.

7. Enter **`$xilinx_tutorial/calc_da/orblk2`** in the component name field and enter **`orblk2`** in the interface name field.
8. Select **`OK`** to execute the command. This saves the symbol as ORBLK2.
9. Close the window containing the symbol.
10. A dialog box appears prompting you to save the changes to ANDBLK2. Since the symbol for ANDBLK2 was saved prior to modifying it for the ORBLK2 symbol, it is not necessary to save changes to the ANDBLK2 symbol. Select **`No`**.

Creating Schematics for ANDBLK2 Symbol

You have created symbols for ANDBLK2 and ORBLK2. The next step is to create schematics for these blocks. The schematics can then be referenced in a higher-level schematic by placing the symbols.

Opening a Schematic Window

1. To open a schematic window, select **`OPEN SHEET`** from the palette. A dialog box appears.
2. Type **`$xilinx_tutorial/calc_da/andblk2`** in the Component Name field, then select **`OK`**. A blank schematic sheet appears.

Adding the First Component to a Schematic

1. From the menu bar, select **`Libraries`** → **`XACT LIB`**. The Xilinx Libraries menu appears.
2. Use the Unified Libraries for new designs. The Obsolete Library is provided for backward compatibility. Select **`Unified Lib`** from the menu.
3. Select the correct library for the device you are targeting, either XC3000 or XC4000. If you select the wrong library, use the PageUp key to go to the top of the library palette menu and click the left mouse button on the Back option. This moves the library menu back up the hierarchy.
4. Choose **`BY TYPE`** from the palette. This option organizes the library parts into categories. The ALL PARTS option displays all

the library parts at once. A menu appears similar to that shown in the figure below.

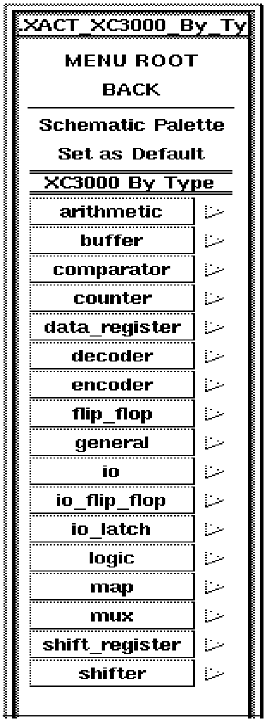


Figure 11-12 XC3000 Library BY TYPE Menu

5. To move up and down in the menu, turn on the scroll bars by moving the cursor into the menu window and selecting **`Right Mouse Button → Show Scroll Bars`**. You can also move up and down using the PageUp and PageDown keys.
6. Click the left mouse button on the **`Set As Default`** option. This option allows you to return to this area and view of the library menu by clicking on the Library icon in the Schematic palette.
7. Choose the **`logic`** category from the BY TYPE menu.
8. Select **and2**.

9. A small dialog box appears on the screen. Move the cursor into the schematic window. The outline of a 2-input and gate appears.
10. Move the symbol outline to the location shown in the following figure and then click the left mouse button to place the object.

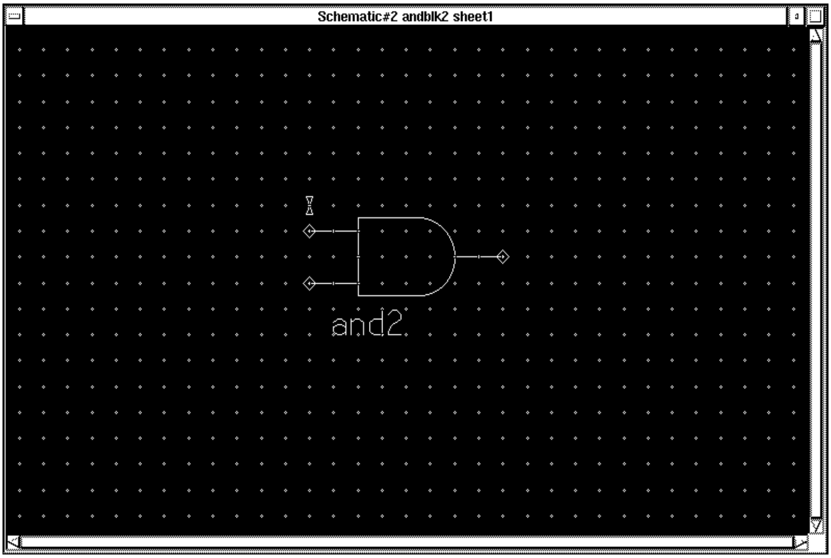


Figure 11-13 Placing a Component

Placing Additional Components

After placing the and2, note that a picture of it appears in the small window in the upper right area of the screen. The last library element selected will appear in this window. To select another component of the same type, move the mouse inside this window, and click the left mouse button. Then move the cursor to the schematic window, position the component, and release the mouse button to place it on the sheet. Using this method, select and place a second and2 symbol as shown in the following figure.

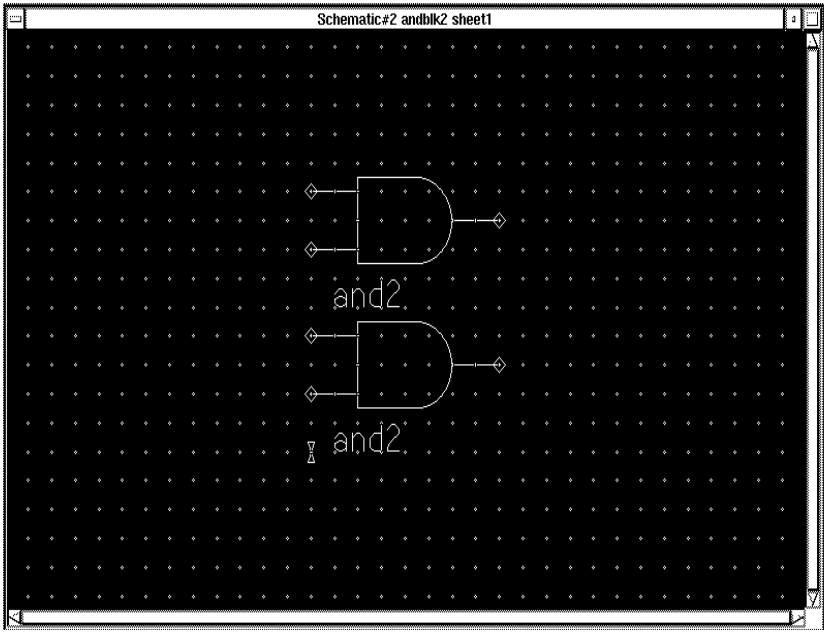


Figure 11-14 Placing a Second Component

Copying a Component

Use the Copy command to add more components by copying a component that already appears on the schematic.

1. Press the F2 function key to ensure that nothing is selected. It is important to use the F2 key before selecting objects because objects selected in previous steps are sometimes not deselected.
2. Move the mouse above and to the left of the two symbols on the sheet.
3. While holding down the left mouse button, move the mouse below and to the right of the two symbols. A white box appears surrounding the two symbols.
4. Release the mouse button to select the objects.

5. Select **Right Mouse Button → Copy**. Alternatively, use stroke 3214789, a stroke in the form of a “C”, to select the copy command. A small dialog box appears at the bottom of the screen.
6. Place the two copied gates above the original two using the left mouse button. If necessary, use the 753 stroke to zoom out. The dialog box disappears after you place the gates.
7. Press Shift - F8 to view the entire schematic. The schematic now looks like the following figure.

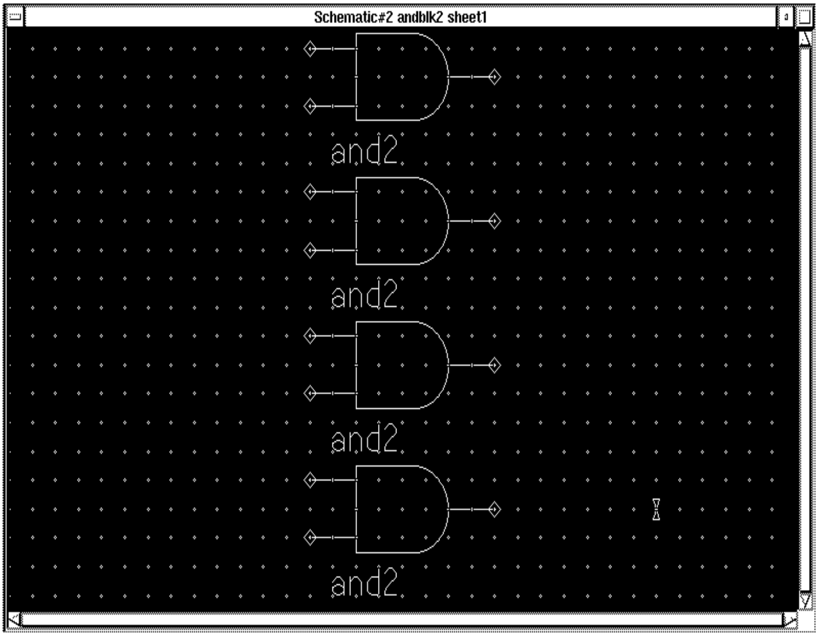


Figure 11-15 Component Placements for ANDBLK2

Moving a Component

If you make a mistake when placing a component, you can use the menu commands to move the component.

1. Use the F2 key to deselect. Select the component by clicking on it with the left mouse button.The component appears highlighted, indicating that it has been selected.

2. Select **`Right Mouse Button → Move`**, or use the stroke 74159. A small dialog box appears.
3. Click the left mouse button to correctly place the component. The dialog box disappears after the component is placed.

Adding Buses to a Schematic

Sometimes it is convenient to draw a set of signals as a bus rather than as several separate wires. It is not necessary to physically connect a bus to the nets that make up the bus. There are several schematics in the Calc design that have short bus segments that are not connected to anything. This is done so that a bus pin can be used to represent the bus on the symbol. A bus must exist on the schematic if a bus pin is to be used for a set of signals.

Add buses to the schematic as follows:

1. After pressing the F2 key, select **`Right Mouse Button → Bus`**: A small dialog box appears, and a white cross appears under the cursor.
2. Draw a bus by clicking the left mouse button to specify the starting point, moving the mouse to a new position, and then clicking the button again to make a bend in the bus or to connect it to a pin. Terminate the bus is by clicking the mouse button in the same place twice. Add the three buses shown in the figure below. You may want to zoom the schematic view out before performing adding the buses.

   If you make a mistake, press the F2 key to deselect everything on the sheet. Then click on the bus segments you want to delete so that they appear highlighted. Press the Delete key and then redraw them correctly.
3. After adding the three buses, press the Escape key to exit the bus adding mode.

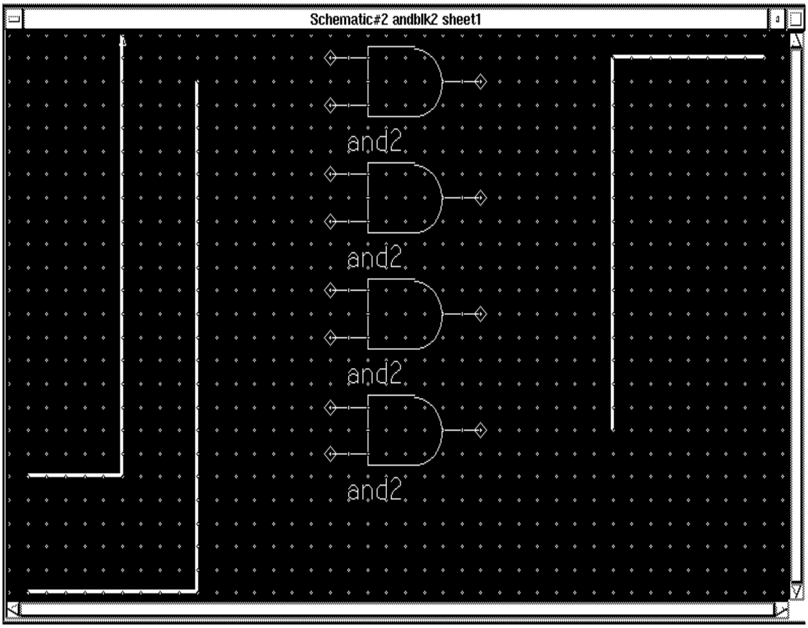


Figure 11-16 ANDBLK2 Schematic with Buses

Adding Nets to a Schematic

Next, nets must be added to attach the appropriate pins on the gates to the buses. You may want to enlarge the view of the components to make it easier to draw the nets.

1. Press the F2 key. Select **`Right Mouse Button → Wire:`** from the ADD menu. A small dialog box appears, and a white cross appears under the cursor.

Note: If the ADD menu does not appear, it may be that something is still selected on the screen, resulting in a different menu appearing on the screen. If this happens, press the F2 key and repeat step one.

2. Move the cursor to the top input pin of the top and2 gate, then click the left mouse button.
3. Move the cursor to the left to a position directly above top of the leftmost bus, so that the wire forms a ninety degree angle with the

bus. Click the left mouse button twice to terminate the wire, which should appear as shown in the figure below.

A bus ripper is inserted automatically between the wire and the bus. A bus ripper defines which bit of the bus is connected to the wire. Automatically inserting bus rippers is referred to as autoripping.

If the bus ripper did not automatically get inserted, make sure that you clicked on the pin first and then on the bus to attach a net between the two. If the net is attached to the bus first, autoripping does not occur. Also, check the Setup menu to make sure that autoripping is turned on. "Set Autoripping Off" should be displayed in the menu to indicate that autoripping is turned on. If "Set Autoripping On" is displayed, select it to turn autoripping on. Also, the $MGC_GENLIB environment variable must be set correctly for the autoripping function.

4. Press the Escape key to exit the wire-adding mode.

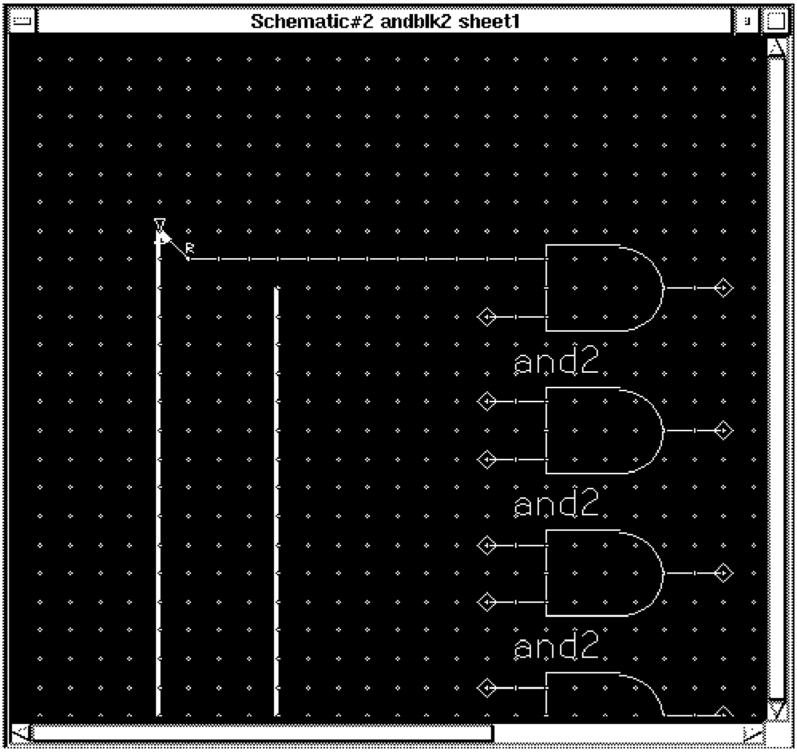


Figure 11-17 Connecting a net

Completing the Net Connections

Add the remaining nets to the schematic as follows:

1. Press the F3 key to execute the Add Wire command.
2. Add the remaining nets as shown in the figure below.

Note: When a wire is properly attached to a symbol pin, the small diamond that specifies the connection point for the pin disappears. If any of the diamonds are still visible, delete the associated net and reattach it.

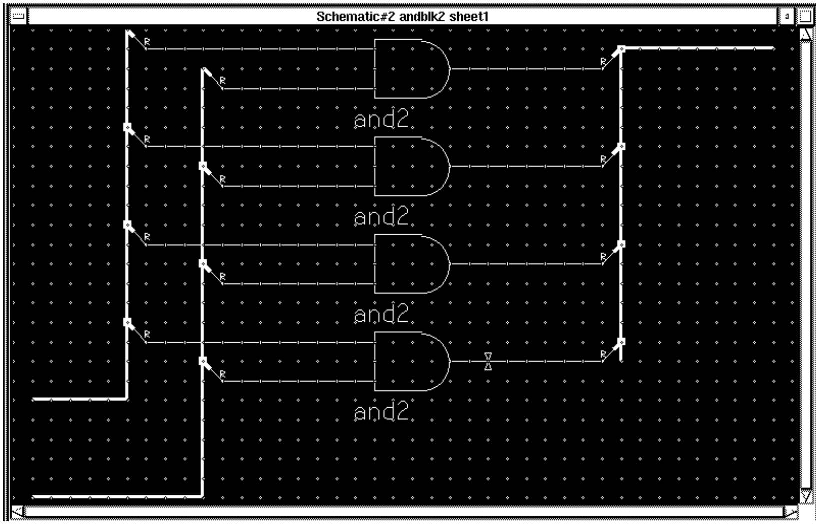


Figure 11-18 ANDBLK2 with all wires and buses connected

Labeling Nets and Buses

The next step is adding labels to the nets and buses. Labeling is the process of identifying a net or a component by assigning a text string to it. It is recommended that you label all nets on the schematic, to simplify debugging and simulation. To specify the bus signals they are related to, all nets that are attached to buses must have a number in parentheses at the end of their names. For example, a net that is bit zero of bus A must be labeled A(0).

1. Press the F2 key to unselect everything on the schematic.
2. Move the mouse above the topmost net and between the symbols and the buses on the right side of the schematic.
3. Press and hold the left mouse button. Drag the mouse downward so that the rectangle covers all four output nets, as shown in the following figure. Release the mouse button. This selects all of the output nets, which then appear highlighted.

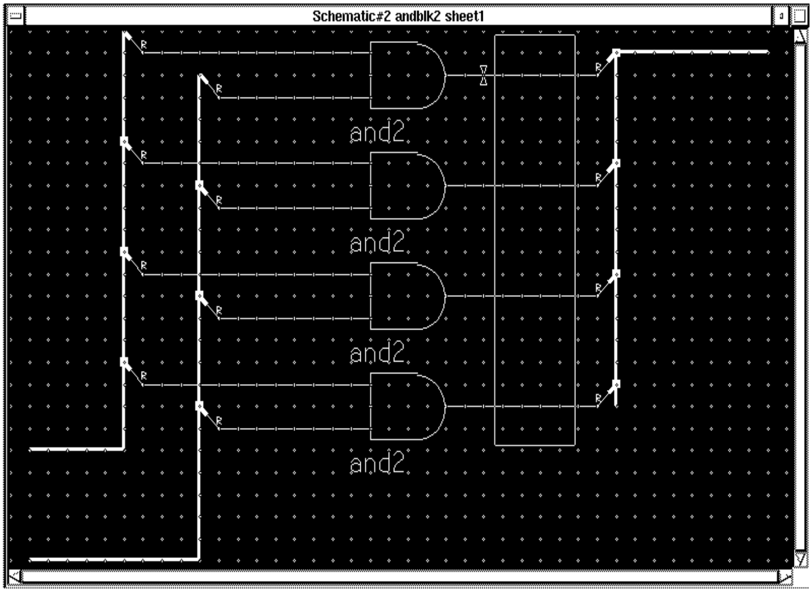
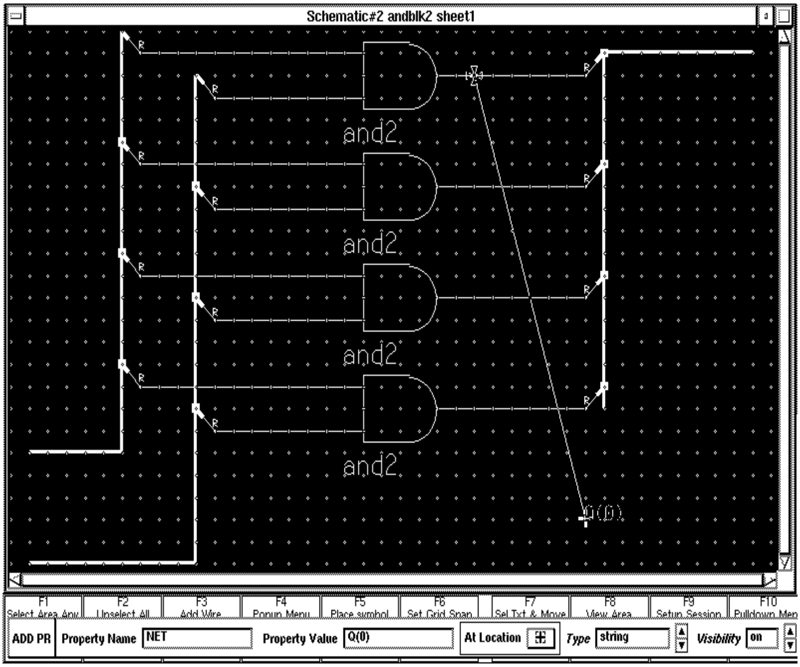


Figure 11-19 Selecting Nets

4. Select **`Right Mouse Button`** → **`Name Nets`**. A crosshair appears next to the topmost selected net, and a small dialog box appears. This indicates that the text typed in the “Property Value” field of the dialog box is used as the name for the net. In Mentor Graphics, a net name is a property with the name “net” and the value is the name of the net. For example, the topmost net will be named Q(0), so type “Q(0)” in the Property Value field of the dialog box and press return.

5. Now, you can place the text on the schematic, although the dialog box does not disappear. When you move the cursor out of the dialog box, the text appears next to it, with a white line indicating the net vertex that is associated with the property, as shown in the figure below. Move the text to the proper position above the net and press the left mouse button to place it.



Property Name NET

Property Value Q(0)

Type string

Visibility on

Figure 11-20 Adding Text

6. The white cross moves down to the next net and a new small dialog box appears.
7. Name the remaining nets. Repeat the appropriate steps to select and label all eight of the input nets as shown in the following figure. Once all the nets are labeled, the small dialog box disappears.

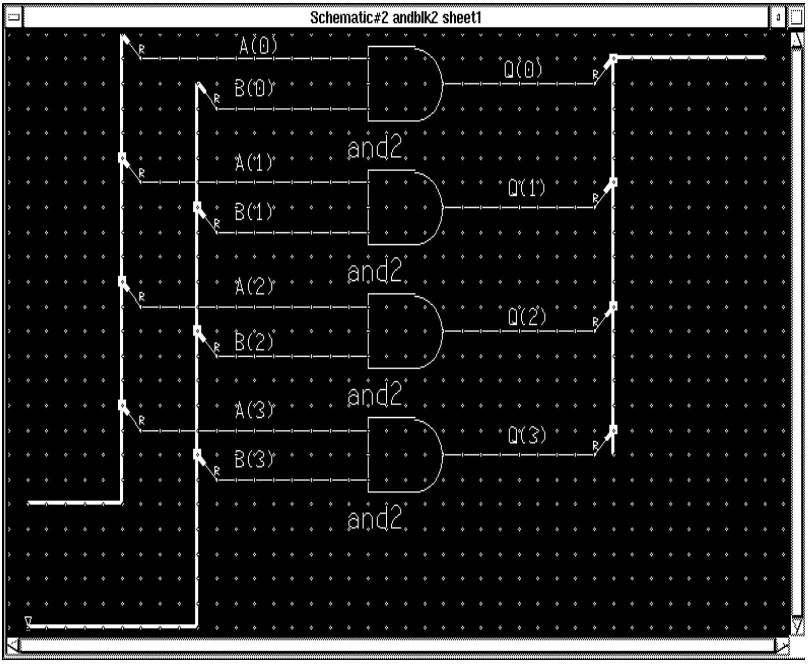


Figure 11-21 Schematic with all Nets Labeled

8. If you incorrectly label a net or bus, move the mouse above the text and press the Shift-F7 keys to execute the **`Text Change Value`** command. Edit the text value that appears in the dialog box and press return.

Adding Ports

Port symbols must be added to nets and buses to define the connectivity between a schematic and its associated symbol. For the ANDBLK2 schematic, all three buses need ports. Input signals are given PORTINs and output signals are given PORTOUTS.

Add ports to the schematic as follows:

1. If the appropriate Unified library is not displayed in the palette, use the menu bar command **`Libraries`** → **`XACT LIB`** to select

it. Then, select the Unified Libraries and the appropriate library for the part being used.

2. If the library is already visible, you may need to choose the BACK option from the top of the library palette to move up to the general library categories. Continue selecting BACK until the ALL PARTS and BY TYPE selections are displayed.
3. Select **`BY TYPE`**, and then choose the **`io`** category.
4. Select the **`portin`** library part from the menu.
5. Place the portin so that the white crosshair is EXACTLY above the left end of the upper input bus, on the left side of the window.
6. Place another portin at the end of the lower input bus, on the left side of the window.
7. Next select a portout symbol from the library and place it at the end of the output bus.
8. Press Shift - F8 to view the entire schematic. The schematic appears as in the following figure.

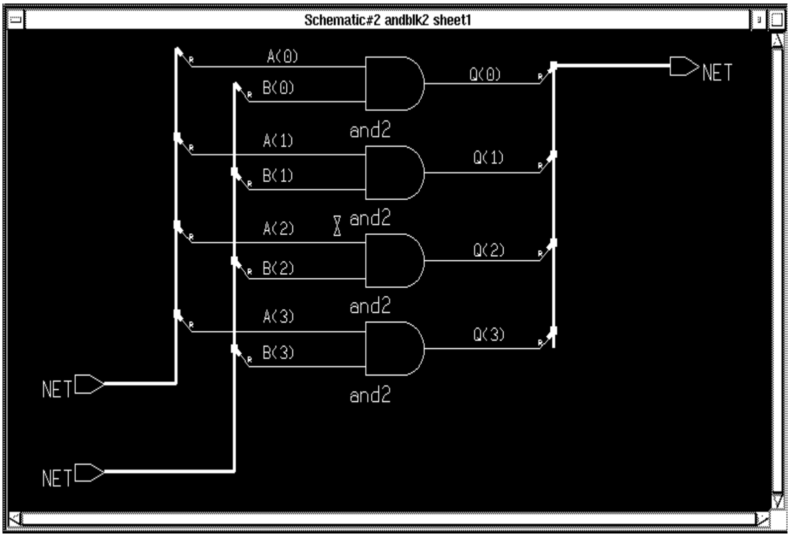


Figure 11-22 Adding Ports

Labeling Buses

Although buses can be labeled using the same method as was used for nets, the addition of the port symbols to the buses has automatically assigned a default name of “NET” to each bus. This simplifies the process since you can modify the existing names rather than add new ones.

1. Press F2 key to unselect everything on the schematic sheet.
2. Move the cursor so that it sits above the NET label on the output bus.
3. Press Shift - F7 to choose the **`Text Change Value`** command. A small dialog box appears.
4. In the New Value field, change the text to Q(3:0).
5. Press return or choose **`OK`** in the dialog box.
6. Repeat this procedure on the two remaining buses, giving them names as shown in the following figure.

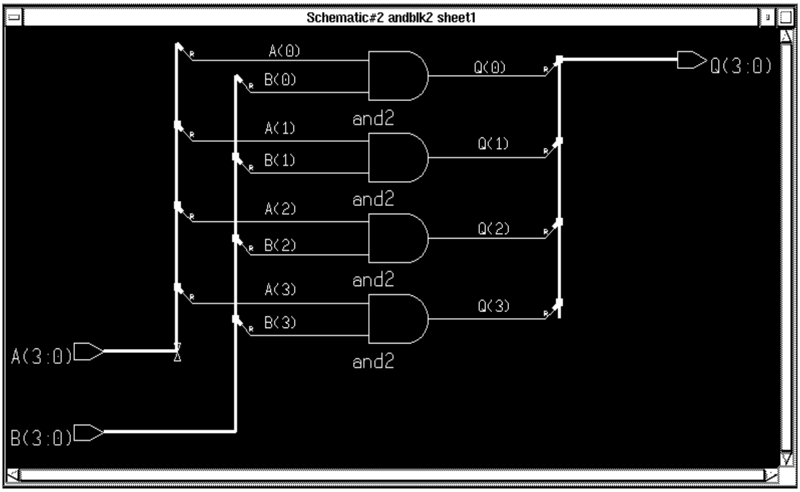


Figure 11-23 Labeling Buses

Defining Bus Ripper Rule Properties

The schematic is almost complete. Although the names on the nets define which bit of the attached bus is associated with the net, it is still necessary to define the bus rippers. Each bus ripper is given a RULE property with a value equal to the number of the bit to which the ripper is attached. For example, the bus ripper attached to net A(0) is given a zero value in order to attach A(0) to bit zero of the bus. Define the bus ripper values using the following method:

1. Press the F2 key to unselect everything on the schematic sheet.
2. Select **`Right Mouse Button`** → **`Other Menus`** → **`Property/Text Menu`** → **`Sequence Text`**. This command allows the numerical sequencing of the modification of text. Since the rippers already have a default value of R for the Rule property, you must change it to the appropriate number.
3. Enter a zero in the Beginning Index Number box, leaving the Prefix and Suffix field empty, then press return. The Prefix and Suffix fields are not necessary because you only want to change the “R”s to 0, 1, 2, and so on. At the bottom of the screen, the message “Click LMB to indicate a text location. Next text will be “0”. This indicates that the next text that the left mouse button is clicked on will be changed to a “0”.
4. Click the left mouse button while positioning the mouse on the “R” attached to the uppermost bus ripper of the bus A(3:0). The “R” changes to a zero. A message indicating that “1” will be the next text appears at the bottom of the screen.
5. Move down the bus A(3:0), clicking on the bus rippers in order.
6. When the fourth bus ripper is reached, click the right mouse button to cancel text sequencing. The four bus rippers are defined as shown in the figure below.

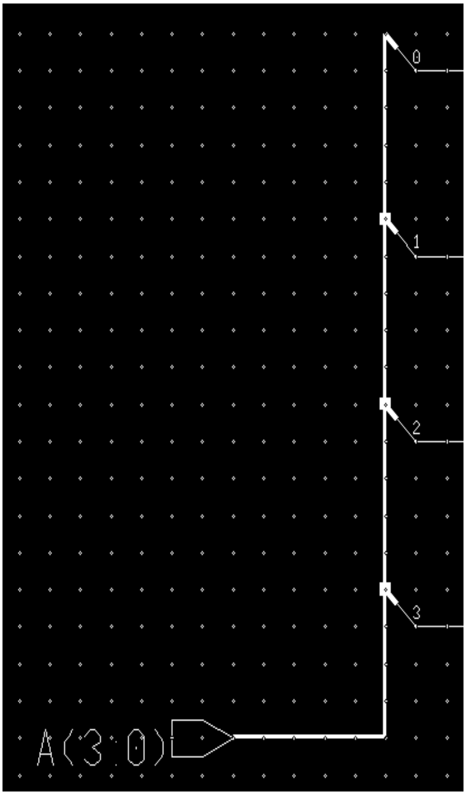


Figure 11-24 Defining Bus Rippers

7. Repeat steps 1-6 for the buses B(3:0) and Q(3:0)

   If mistakes are made defining the bus rippers, you can change the number using the same method (Shift - F7) used to edit net and bus names. Make sure they are defined with numbers that match the attached nets. A bus ripper definition problem can be difficult to resolve.

Saving the Schematic

The schematic is now complete. Check and save the schematic as follows:

1. Select **`Check`** → **`Sheet.`** The text window that appears may contain warnings about unnamed or dangling net vertices. These warnings can be safely ignored, but if any other warnings or errors occur, recheck the schematic against the figure below. The check sheet window is also linked to the schematic window. Any net, vertex, or instance names can be highlighted in the check sheet window by clicking the left mouse button on it. The corresponding net, vertex, or instance on the schematic is highlighted. This is useful for relating an error message in check sheet to the schematic.

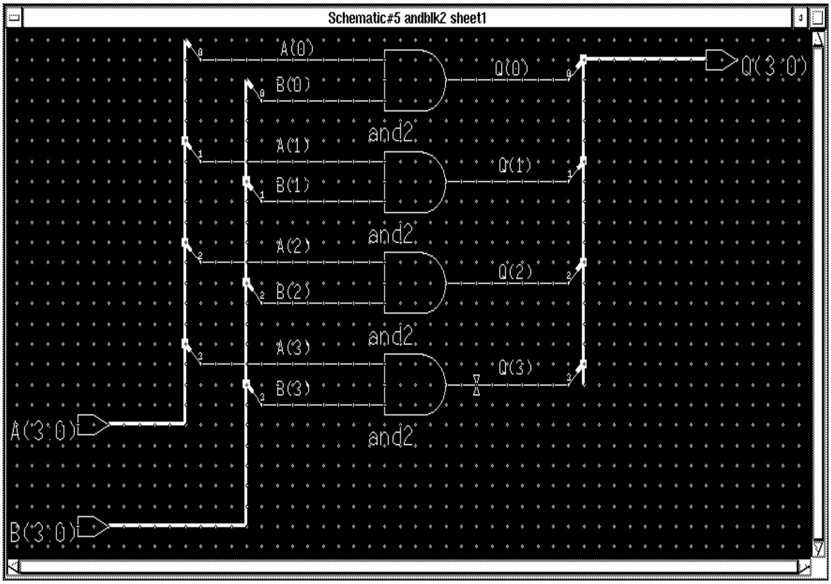


Figure 11-25 Completed ANDBLK2 Schematic

2. Once all schematic errors have been corrected, check the design again if necessary, and close the check sheet text window. Select **`File`** → **`Save Sheet`** from the menu bar to save the schematic.

Creating Schematics for ORBLK2 Symbol

The ORBLK2 schematic is similar to the ANDBLK2 schematic. To create schematics for the ORBLK2 symbol, you can use the ANDBLK2 schematic and simply replace the four and2 gates with or2 gates.

1. Press F2 key to unselect everything on the ANDBLK2 schematic.
2. Display the BY TYPE library menu. Select the **`logic`** category.
3. Press and hold the left mouse button and move the mouse to create a rectangle to include part of all four and2 gates, as shown in the following figure. It is not necessary to box the entire gate to select it. Do not include any part of the attached nets in the rectangle.

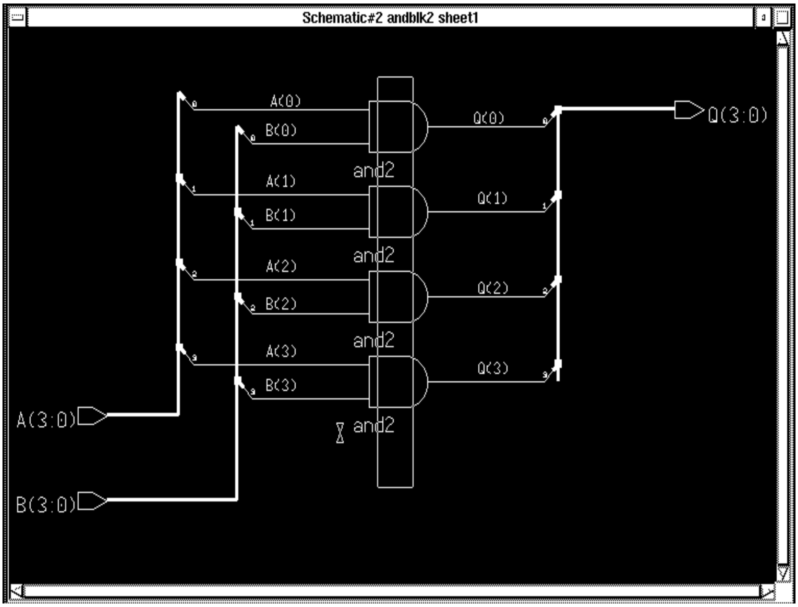


Figure 11-26 Selecting Gates

4. When the rectangle is positioned correctly, release the left mouse button to select all four and2 gates.

5. Select **Right Mouse Button → Replace → From Library Menu.** A message appears at the bottom of the screen requesting that you select the replacement library part from the menu.
6. Use the PageUp and PageDown keys to scroll the component list. Select the or2 component.The four and2 gates are replaced with or2 gates. The ORBLK2 schematic is complete.
7. Select **Check → Sheet** from the menu bar. The check program refers to the ANDBLK2 schematic, since this was modified to create the ORBLK2 schematic.
8. Close the text window containing the results of check sheet.
9. Select **File → Save Sheet As...** A dialog box appears.
10. Type **$xilinx_tutorial/calc_da/orblk2** in the Component name field. Leave all other fields blank.
11. Press return to save the schematic.

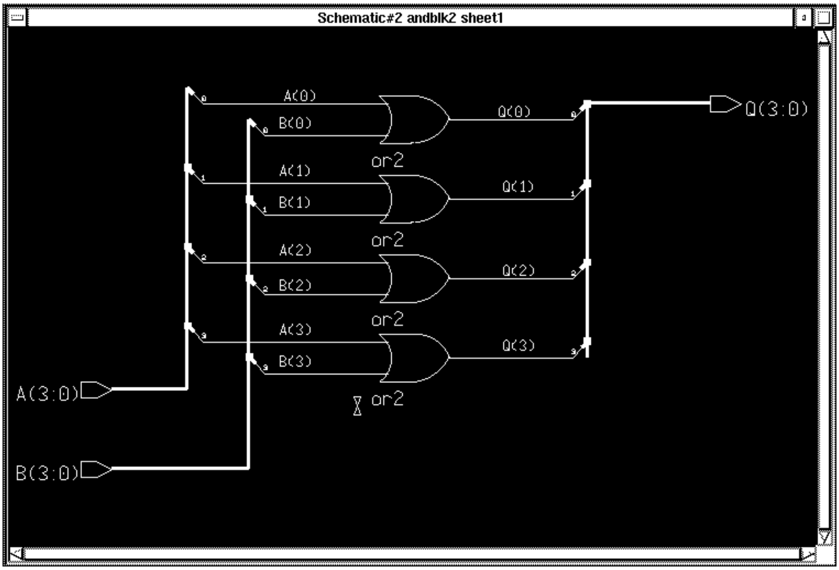
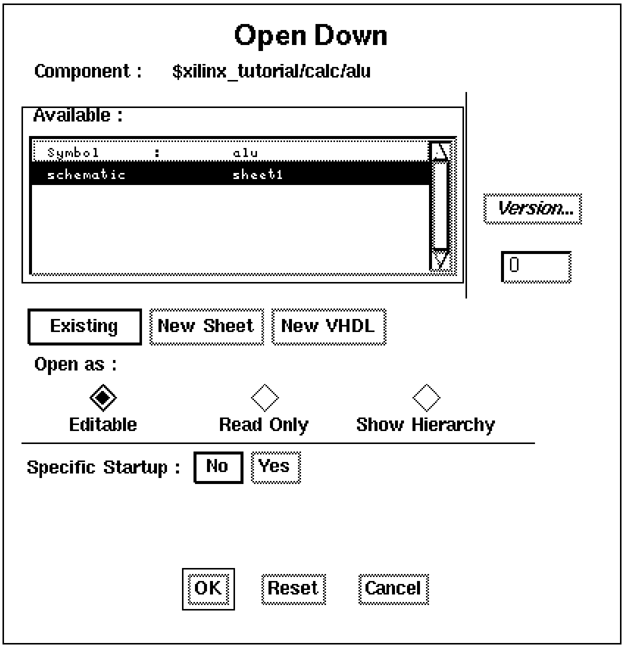


Figure 11-27 Completed ORBLK2 Schematic

Completing the ALU Schematic

So far you have created symbols for ANDBLK2 and ORBLK2. You have also created underlying schematics for these symbols. The next step is to place the symbols in the ALU block schematic.

1. Close the only open window, which is the modified ANDBLK2 schematic, using the button in the upper left corner of the window. A dialog box appears asking whether to save the changes to the schematic. Select **No**, since the ANDBLK2 schematic was saved earlier, but then modified for use as the ORBLK2 schematic.
2. Choose **OPEN SHEET** from the session palette.
3. Press the Navigator... button. A navigator window appears. If necessary, change directories to the $xilinx_tutorial/calc_da directory using the up arrow to move up one directory level and double-clicking folders to push into them. Then, select the Calc design, which is represented by a folder with a “c” on it and with the name “calc” next to it. The “c” specifies that it is a component, and not just a directory.
4. Press return or select **OK** from the navigator window. The Component Name field of the OPEN SHEET dialog box is automatically back-filled with “$xilinx_tutorial/calc_da/calc”.
5. Press return or select **OK** from the OPEN SHEET dialog box. The top level Calc design appears in a window. Press Shift - F8 to view the entire schematic, if necessary.
6. Press F2 key to unselect everything on the schematic.
7. Select the ALU symbol.
8. The additions you need to make are all in the ALU schematic, so choose **File → Open Down** from the menu bar using the left mouse button. The dialog box shown in the following figure appears.
9. You must select whether to modify the symbol or the schematic for ALU. Select the line, schematic sheet1, with the left mouse button. It now appears highlighted, as shown in the figure below.
10. Press return or select **OK**. A second schematic window appears; it contains the ALU schematic.



Open Down

Component : $xilinx_tutorial/calc/alu

Available :

Symbol	:	alu
schematic		sheet1

Version... 0

Open as : ◆ Editable ◇ Read Only ◇ Show Hierarchy

Specific Startup : [✓ No] Yes

Figure 11-28 Open Down Dialog Box

Placing User-Created Components

The ANDBLK2 and ORBLK2 symbols can now be placed on the schematic as shown in the figure below. The symbols can be placed using the same procedure used to place the and2 gate from the Xilinx libraries when you created the ANDBLK2 schematic.

1. Use the F8 key to zoom into the empty area near the center of the schematic, between the XORBLK2 and ADSU4 symbols.
2. Press the F2 key to ensure that nothing is selected.
3. Choose **`Right Mouse Button → Instance → Symbol by Path...`** A dialog box appears.
4. Use the Navigator button in the dialog box to select the $xilinx_tutorial/calc_da/andblk2 component, or type the name in the Component Name field of the Add Instance dialog box.

5. Press return or select **OK** to execute the command.
6. Move the cursor to the correct location as shown in the figure below.

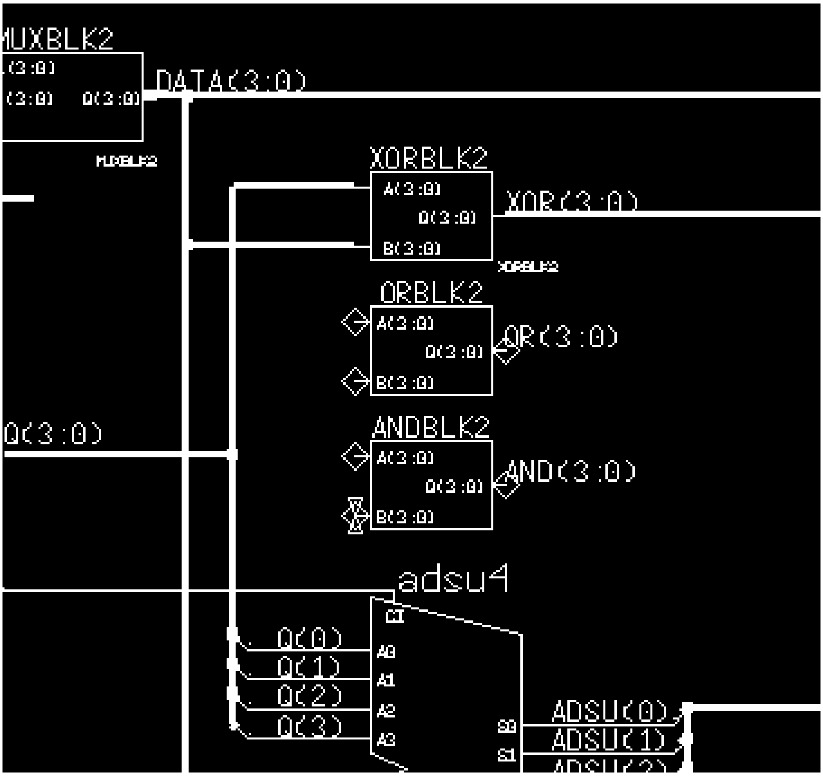


Figure 11-29 Adding ANDBLK2 and ORBLK2 to ALU Schematic

7. Press the left mouse button to place the component.
8. Follow the same procedure to add the ORBLK2 symbol. Refer to the ALU schematic in the figure above for proper placement. If you make a mistake when placing a component, select it (after pressing F2 key) and choosing **`Right Mouse Button → Move`** to reposition it.

Placing Library Components

The next step in the tutorial is to add the fd4ce component to the ALU schematic. The fd4ce component is available in the Xilinx Unified Libraries and consists of four flip-flops with a clock enable.

Note: This component is available in both the XC3000 and XC4000 libraries.

1. Use the Shift -F8 to display the entire ALU schematic. Use the F8 key to zoom into the open area in the lower right-hand corner.
2. Select **`Libraries`** → **`XACT LIB`** from the menu bar.
3. Select the Unified Libraries and the appropriate family library using the left mouse button.
4. Choose **`BY TYPE`**→ **`flip_flop`** → **`fd4ce`** from the Library menu. Move the cursor into the schematic window; an outline of the fd4ce component appears.
5. Move the component to lower right corner of the schematic, approximately to the location shown in the figure below.
6. Press the left mouse button to place the component.

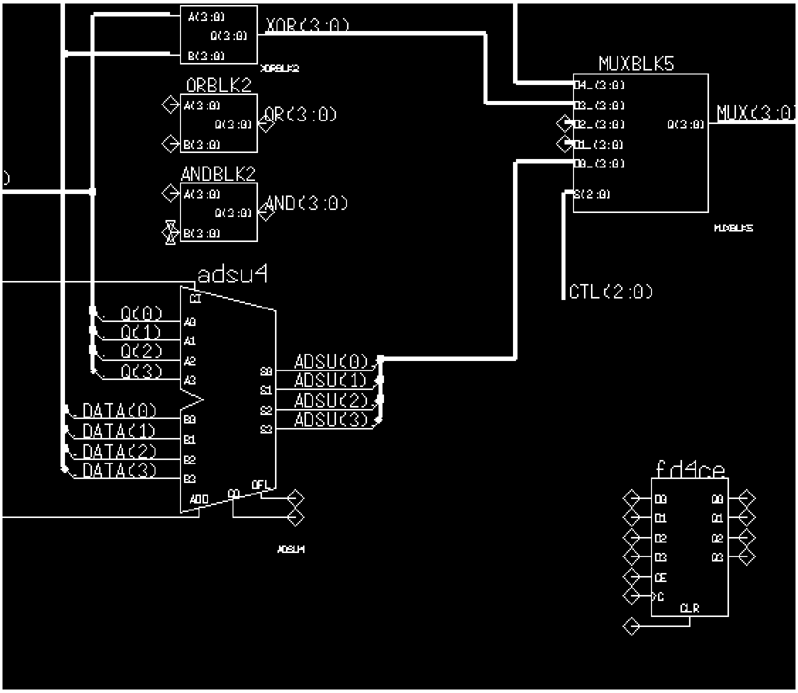


Figure 11-30 Adding fd4ce to ALU Schematic

Adding Nets, Buses, Ports and Labels

FD4CE

Next complete the addition of the fd4ce symbol by adding nets, buses, and labels as follows:

1. Add the necessary nets and buses to complete connections for fd4ce as you did for the previous schematic. The figure below displays the labeled nets and buses for fd4ce.
2. Add ports to the nets and buses attached to the fd4ce, as shown in the figure below.
3. Change the default “NET” properties to the proper names using the Shift-F7 key, as shown in the following figure.

4. Modify the bus ripper RULE properties appropriately, as described in a previous step.

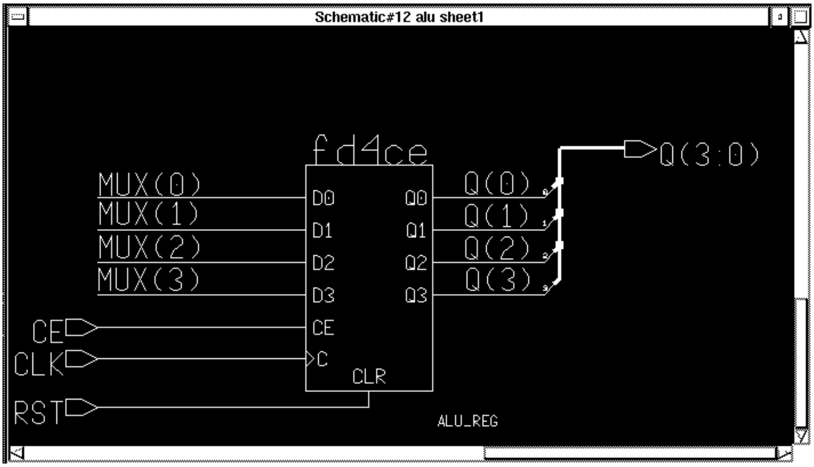


Figure 11-31 Nets, Buses, and Ports for fd4ce

ANDBLK2 and ORBLK2

Next, complete the addition of ANDBLK2 and ORBLK2 to the ALU schematic.

1. Add the necessary buses to complete connections for ANDBLK2 and ORBLK2. The figure below displays the labeled nets and buses for ANDBLK2 and ORBLK2.
2. Use the figure below to name the added buses, as described in the “Labeling Nets” section of this chapter. You only need to label the output buses of the two components.

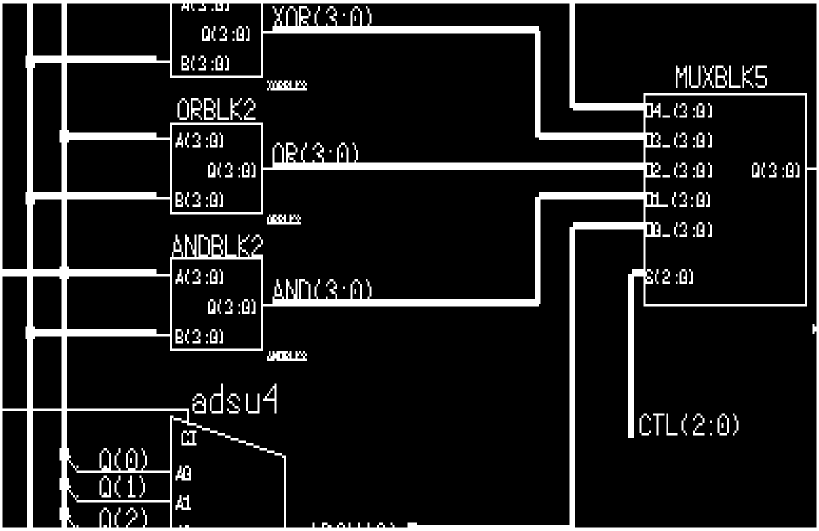


Figure 11-32 Nets, Buses and Labels for ANDBLK2 and ORBLK2

Adding Labels to Components

It is important to add labels to components. Error and warning messages often reference component labels, and labels also appear in simulation netlists. Also, net names at lower levels of hierarchy are referenced using the following format:

...component_label/component_label/net_label

In the ALU schematic, labels have already been added to the MUXBLK2, XORBLK2, and MUXBLK5 blocks.

To add a label to the ANDBLK2 placement, follow these steps.

1. Press the F2 key to unselect everything.
2. Use the left mouse button to select the ANDBLK2 symbol.
3. Select **`Right Mouse Button`** → **`Properties`** → **`Add`**. A dialog box appears.
4. In the window labeled “Existing property Name”, choose the INST property with the left mouse button. It appears highlighted.

5. In the Property Value field, type **ANDBLK2**, then press return or choose **OK**.
6. Move the text to position it as shown in the following figure. Click the left mouse button to place the text.
7. Label the ORBLK2 symbol the same way using the label, ORBLK2, as shown in the following figure.
8. Give the fd4ce component the label, ALU_REG.

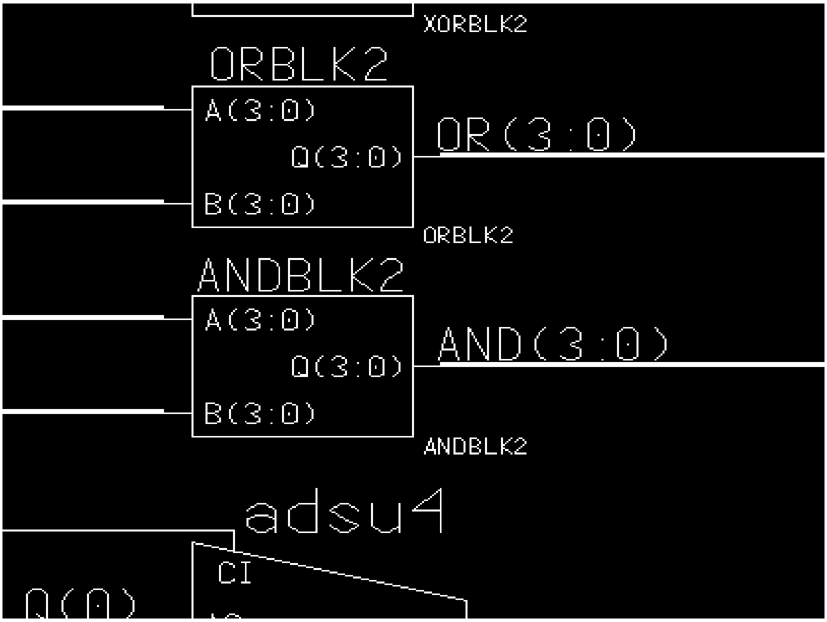


Figure 11-33 Adding Component Labels to ALU Schematic

9. The completed ALU schematic is shown in the following figure.

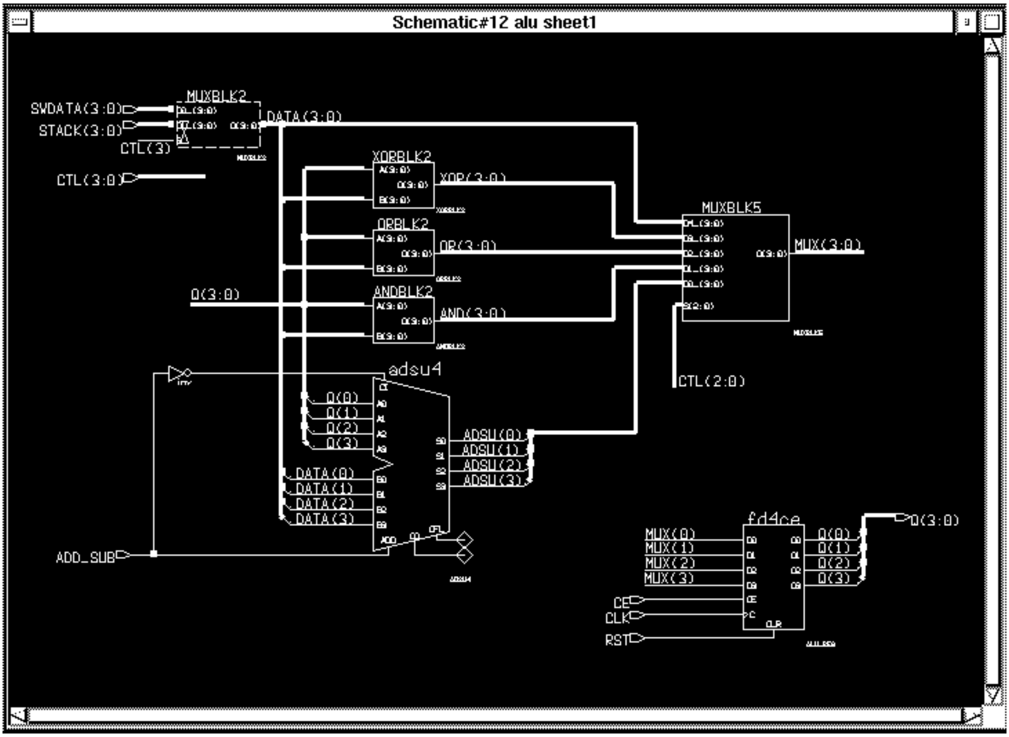


Figure 11-34 Completed ALU Schematic

Saving the ALU Schematic

Check the schematic. If errors occur, resolve them and then check and save the schematic.

Exploring Xilinx Library Elements

The Xilinx libraries contain three types of elements. Primitives are basic logic elements such as the and2 and or2 gates that you previously placed in ANDBLK2 and ORBLK2. Soft macros are schematics created by combining primitives and other soft macros. Relationally Placed Macros (RPMs) are soft macros that contain placement information. RPMs are currently only available in the XC4000 library.

All three types of library elements are placed on a schematic in exactly the same way.

Viewing a Xilinx Soft Macro Schematic

Soft macro schematics are schematics such as you might make for your own designs. In fact, you can load one of these schematics and use the File Save As command to save it under another name, and then edit this new schematic to customize it to your needs.

Open the schematic underneath the fd4ce symbol as follows:

1. Press the F2 key to unselect everything.
2. Select fd4ce with the left mouse button.
3. Select **`File` → `Open Down`** from the menu bar. A dialog box appears. Select the schematic sheet and choose **`OK`**. As shown in the following figure, fd4ce consists of four fdce symbols.

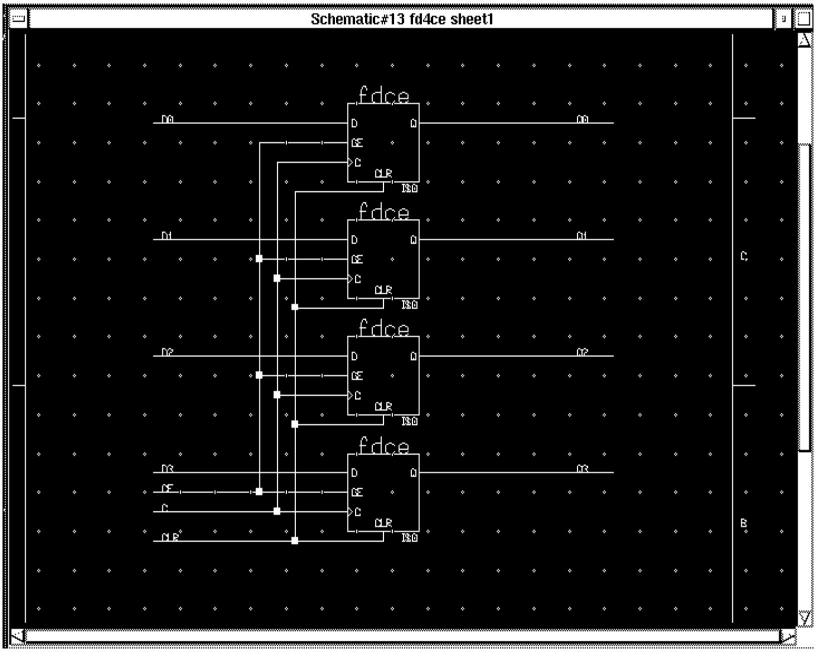


Figure 11-35 fd4ce Schematic from XC3000 Library

Viewing a Xilinx Library Primitive

The lowest-level elements in the Xilinx libraries are the logic primitives. Only logic primitives appear in the eventual netlist produced from the design. For example, the flattened netlist for the Calc design does not contain a reference to the fd4ce symbol on the ALU schematic, but does contain the four fdce instances in the fd4ce schematic. Simulation models, which consists of Mentor Graphics gen_lib primitives, are found below Xilinx library primitives.

1. Use the left mouse button to select one of the fdce placements.
2. Select **`File`** → **`Open Down`** from the menu bar. A dialog box appears. Select the schematic sheet and choose **`OK`**. The schematic shown in the following figure consists of simulation modeling parts from the gen_lib library. At the lowest level, all soft macros and RPMs are made up of these. You never need to modify or copy schematics at this level.
3. Close the fdce schematic and reselect the fd4ce schematic.

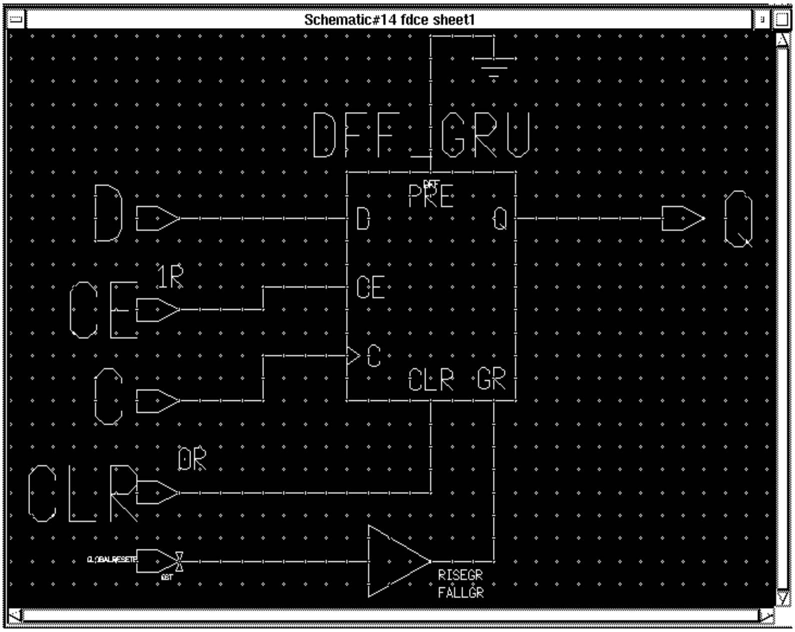


Figure 11-36 fdce Schematic

Viewing a Xilinx RPM (XC4000 Family Only)

Note: The following description of RPMs contains detailed information on the XC4000 architecture. Refer to *The Programmable Logic Data Book* for more information on the XC4000 CLB structure and fast carry logic.

If your design is not targeted for the XC4000 family, read this section, but do not perform any of the commands. Continue the tutorial with the next section, “Opening the Calc Schematic.”

The ALU contains a component from the Xilinx library, adsu4, which is a four-bit wide adder/subtracter. If your design is targeted for the XC4000 library, this schematic is implemented as a Relationally Placed Macro (RPM). If your design is not targeted for the XC4000 library, adsu4 is implemented without this placement information.

RPM schematics are schematics such as you might make for your own designs. In fact, you can load one of these schematics and use the File Save As command to save it under another name. You can then edit this new schematic to customize it to your needs.

Elements placed in the adsu4 RPM schematic include CY4 components and FMAPs. The CY4 symbol gives you the ability to specify fast carry logic functionality from the schematic. Fast carry logic is a hardware feature in XC4000 parts that allows very fast arithmetic-type functions.

The FMAPs map logic functions to function generators in Configurable Logic Blocks (CLBs), which are arranged in a rectangular grid in the die. Both CY4 symbols and FMAP symbols have RLOC attributes. RLOCs are attached to the symbols that assign relative locations to the CLBs. You can use carry symbols as well as FMAPs and other mapping components in your own schematics. However, knowledge of them is not necessary to use RPMs. Only expert users should create macros containing carry logic and FMAPs. For a description of these components, see the *XACT Libraries Guide*.

Push into the adsu4 schematic as follows:

1. Press F2 key.
2. Select adsu4.
3. Open the schematic underneath adsu4.

4. Use the F8 key (or stroke 1 59) to zoom into the upper portion of the schematic as shown in the following figure.
5. Press F2 key to unselect everything.
6. Select the FMAP component in the upper right corner.

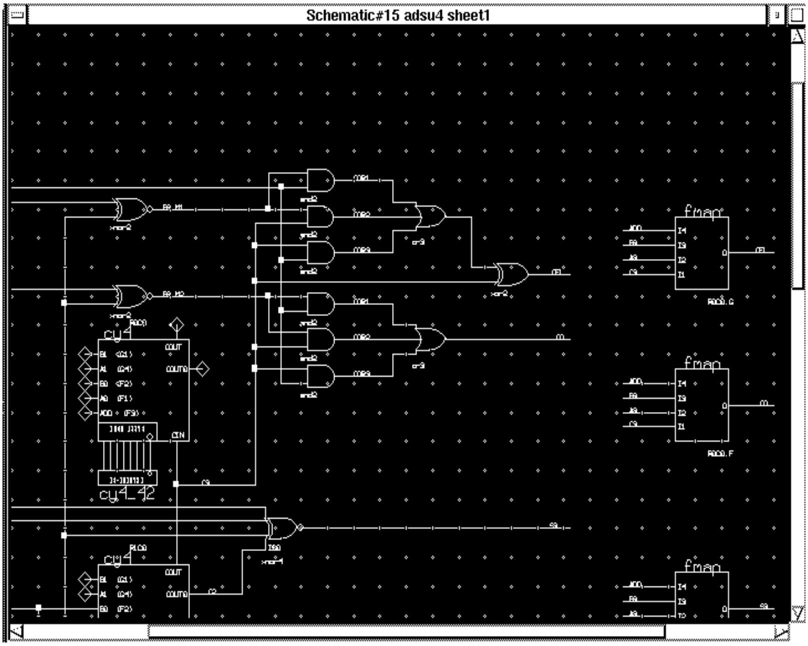
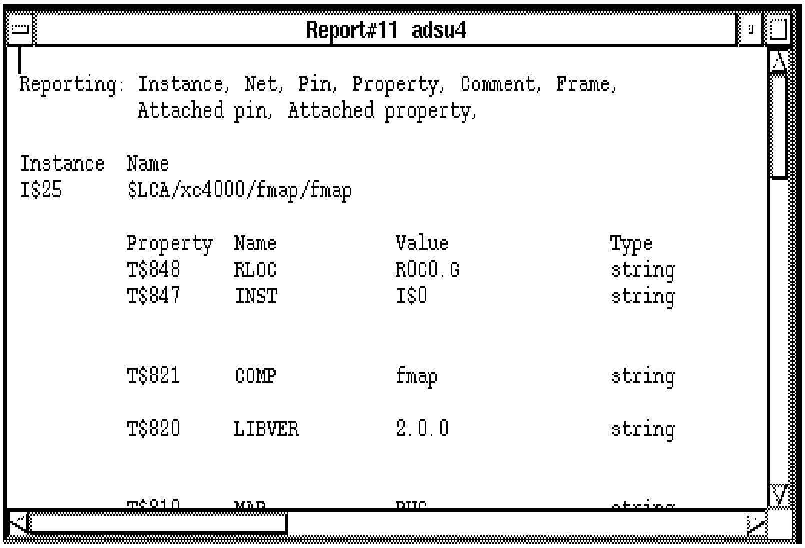


Figure 11-37 Upper portion of the ADSU4 RPM Schematic

7. Select **`Report`** → **`Object`** → **`Selected`** → **`All`**. A text window appears displaying the attributes on the symbol, as shown in the figure below. The RLOC attribute is set to R0C0.G, indicating that this function is mapped to the G function generator of the upper-left corner (row zero, column zero) CLB in the RPM. RPM origins are in the upper left-hand corner.
8. Close the text window to return to the adsu4 schematic window.
9. Use the scroll bars on the sides of the window to pan around the schematic and look at the RLOCs. Note that logic is mapped to three CLBs, designated as R0C0, R1C0, and R2C0. Therefore, this

RPM uses three CLBs that are arranged in a column. Information on the number of CLBs used and the shape of the logic block is available for each RPM in the *XACT Libraries Guide.* Note that these locations are relative, not absolute. The macro is not defined as placed in the uppermost CLB in the left most column. Regardless of what the RPMs absolute location, the logic associated with the FMAP with the location R0C0 is always at the top, R1C1 is in the CLB directly below, and so on.

10. Close the adsu4 schematic and return to the ALU schematic.



```
Reporting: Instance, Net, Pin, Property, Comment, Frame,
           Attached pin, Attached property,

Instance  Name
I$25      $LCA/xc4000/fmap/fmap

          Property  Name          Value           Type
          T$848     RLOC          R0C0.G          string
          T$847     INST          I$0             string


          T$821     COMP          fmap            string

          T$820     LIBVER        2.0.0           string


          T$819     MAP           PUC             string
```

Figure 11-38 RLOC Attribute on FMAP Component

Opening the Calc Schematic

Close all open schematic or symbol windows except for the top-level Calc schematic window. If the Calc window is closed, open it. The Calc schematic appears on the screen.

Using the XC3000 Oscillator

If your design is not targeted for the XC3000 family, read this section, but do not perform any of the commands. Continue the tutorial with the next section, "Using the XC4000 Oscillator."

The XC3000/XC4000 and XC3000 demonstration boards have a built-in RC circuit for clock generation for the XC3000 family part. The OSC_3K block contains an oscillator that connects to that circuit. The frequency of the output varies with processing, so it is not suitable for applications that require a very precise clock. For this design, however, the oscillator is adequate.

The output of the oscillator circuit in OSC_3K is routed through a global clock buffer before being passed to the rest of the device. There are several reasons for using one of the two global buffers available in the XC3000 family of devices. The global clock buffers drive dedicated routing resources that can reach any clock pin in the device very quickly with minimal skew. In addition, using the dedicated clock nets frees up other general purpose programmable interconnect for other signals.

The GCLK symbol at the right side of the OSC_3K schematic in the figure below is the Xilinx primitive for the XC3000 global clock buffer. Another available global clock buffer is ACLK, the alternate clock buffer. In general, you will use at least one clock buffer (GCLK or ACLK) in every clocked XC2000 or XC3000 design. The Calc design uses the global clock buffer, GCLK. Use GCLK for the highest-priority clock net (the largest fan-out or fastest clock net) and use ACLK for the second-highest-priority clock.

Push into the OSC_3K schematic as follows:

1. Press F2 key to unselect everything.
2. Select the OSC_3K symbol at the lower left.
3. Use **`File`** → **`Open Down`** to open the schematic underneath OSC_3K. The OSC_3K schematic appears as shown in the figure below.
4. Close the OSC_3K schematic.

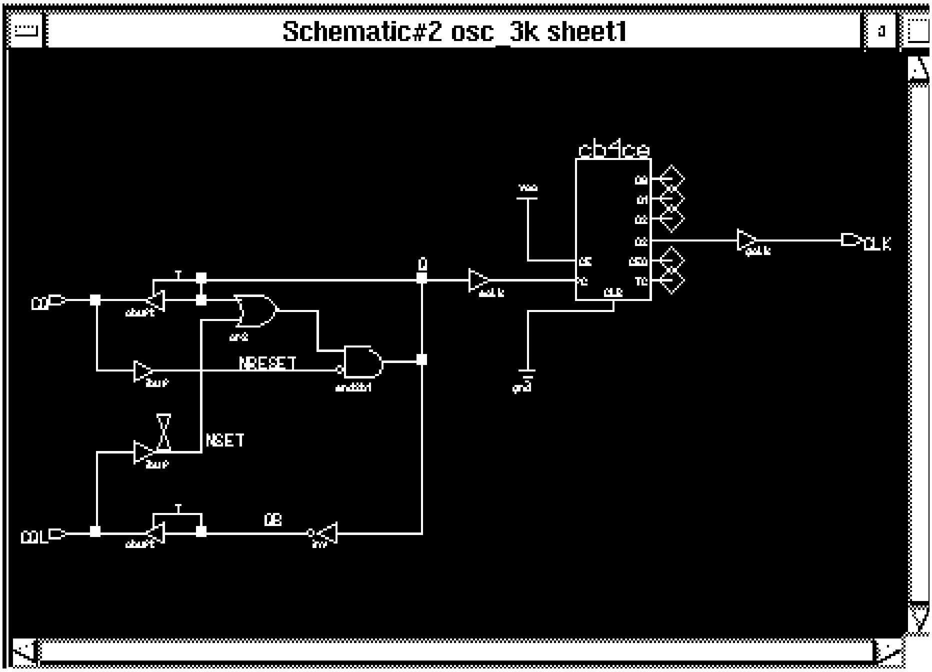


Figure 11-39 OSC_3K Schematic

Using the XC4000 Oscillator

If your design is not targeted for the XC4000 family, read this section, but do not perform any of the commands. Continue the tutorial with the next section, "Inverting Output Display Signals."

The XC4000 family devices contain an on-chip clock generator, which makes it unnecessary to use an external circuit for this purpose. The on-board clock circuitry is not precise, but is suitable for designs that do not need a highly accurate clock, such as the Calc design. If your design is targeted to an XC4000 family device, you must replace the OSC_3K component with one that uses the XC4000 family oscillator and the XC4000 family clock buffers. The OSC_3K symbol is replaced with the OSC_4K symbol.

The OSC_4K schematic contains an XC4000 library part, OSC4. This symbol represents the on-chip oscillator that generates nominal clock frequencies of 8 MHz, 500 KHz, 16 KHz, 490 Hz, and 15 Hz. The Calc design uses the 15-Hz output from this component when targeted for XC4000 family designs. The clock output from OSC4 is buffered through a BUFGS global clock buffer.

XC4000 family devices have eight on-chip clock buffers, one BUFGP (primary global buffer) and one BUFGS (secondary global buffer) in each corner of the device. Although it is possible to use them for other purposes, BUFGPs are best used to route externally-generated clock signals. BUFGSs have more flexibility, and can be used to route any large fan-out net, even if it is internally sourced. The Calc design uses a BUFGS because the clock is generated internally by the on-chip oscillator. See the *XACT Libraries Guide* and the *The Programmable Logic Data Book* for more information on global clock buffers for Xilinx devices.

Replace the OSC_3K with OSC_4K and push into the OSC_4K schematic as follows:

1. Press F2 key.
2. Select the OSC_3K instance on the schematic.
3. Select **`Right Mouse Button → Replace → Other.`** A dialog box appears.
4. Type **`$xilinx_tutorial/calc_da/osc_4k`** in the Component Name field, or select the component using the Navigator button. The OSC_3K symbol is replaced with the OSC_4K symbol. Note that the two pad inputs, CQ and CQL, are now loadless. These pads and their associated nets can be deleted if desired. If they are not deleted, they are optimized out of the netlist later.
5. Confirm that the OSC_4k symbol is the only object selected. Select **`File → Open Down`** and open the schematic sheet under OSC_4K. The OSC_4K schematic appears as shown in the following figure.
6. Close the OSC_4K schematic and return to the Calc schematic.

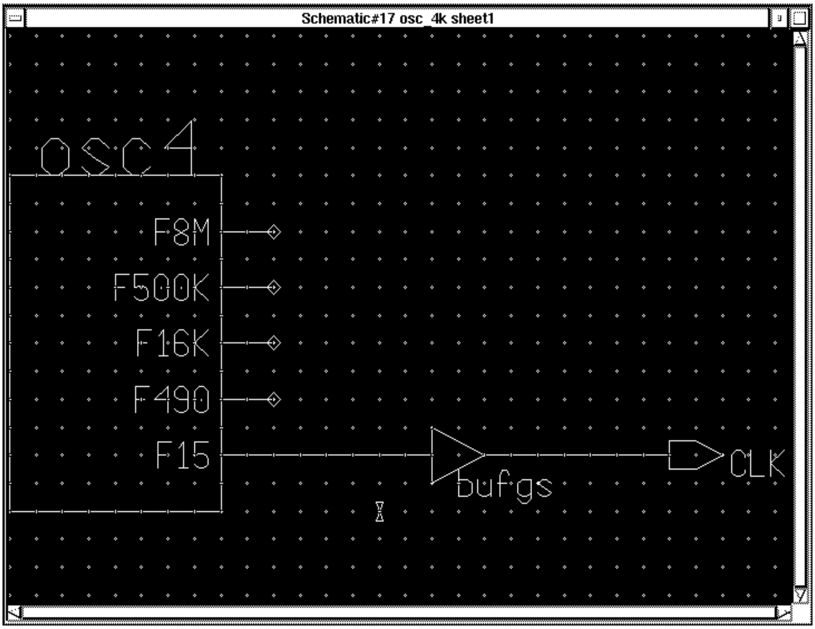


Figure 11-40 OSC_4K Schematic

Inverting Output Display Signals

Note: XC3000 Demonstration Board Only.

The XC3000/XC4000 demonstration board and the XC4000 demonstration board are both designed so a low signal on an output turns on the display element. Therefore, the component that drives the display, 7SEG_DEC_INV, produces inverted sense outputs. The Calc schematic is set up for this configuration as the default.

The XC3000 demonstration board has only a single LCA socket containing an XC3000 family part in a PC68 package. It is designed so that a display element is turned on when the input is driven HIGH. If you plan to target the design to an XC3000 demonstration board, the 7SEG_DEC_INV component on the Calc schematic must be replaced with its non-inverting equivalent. If you plan to download to an XC3000 demonstration board, continue with the commands in this section. If not, go to the next section, “Controlling LCA Layout from the Schematic.”

To make the output signals compatible with the XC3000 demonstration board, make the following changes to the Calc schematic.

1. Unselect everything using F2 key.
2. Click the left mouse button on the 7SEG_DEC_INV symbol.
3. Select **Right Mouse Button → Replace → Other..**. A dialog box appears.
4. Type **$xilinx_tutorial/calc_da/7seg_dec** in the field labeled Component Name, then press return or choose **OK**. The 7SEG_DEC_INV symbol is replaced by the symbol for 7SEG_DEC. This changes the outputs to the seven-segment display to the correct polarity. The outputs driving the led bar must still be changed.
5. Unselect using the F2 key.
6. Select the Library icon in the palette. Since XC3000 users should have set the XC3000 library as the default earlier, this library appears in the palette, organized by type.
7. Choose **buffer** from the menu.
8. Using the left mouse button, select the four inverters attached to the STACK(3:0) bus, as shown in the following figure.
9. Select **Right Mouse Button → Replace → From Library Menu.** A message appears at the bottom of the Design Architect window requesting that you select the replacement part from the library window.
10. Select the buf symbol from the library menu.The inverters are replaced by buffers. Extraneous buffers in the design does not adversely affect its implementation. Any unneeded buffers are automatically removed from the netlist. The schematic is now in a state suitable for download to an XC3000 demonstration board.

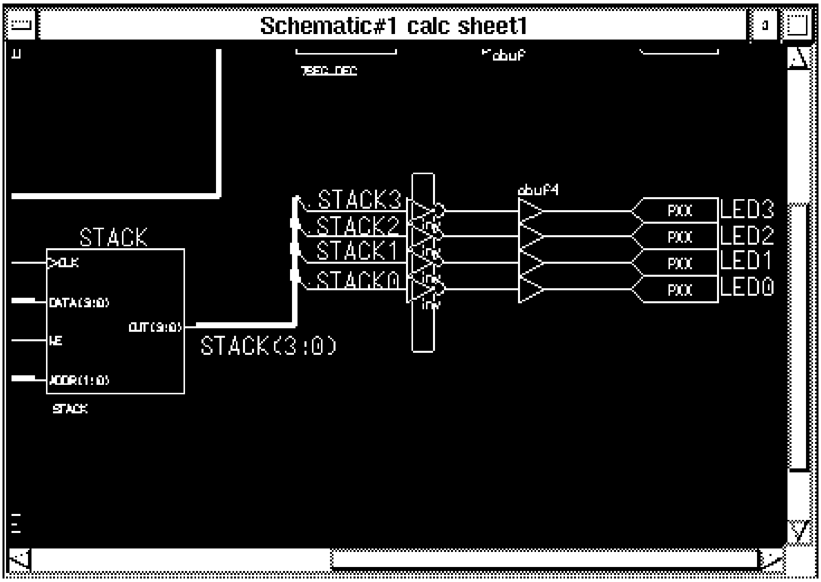


Figure 11-41 Selecting Inverters

Controlling FPGA Layout from the Schematic

Assigning Pin Locations

It is highly recommended that you let the automatic placement and routing programs, PPR and APR, define the pinout. Pre-assigning locations to the I/Os can sometimes degrade the performance of the place and route tools. However, it is usually necessary, at some point, to lock the pinout of a design so that it can be integrated into a board design. The initial pinout should be defined by running the place and route tools without pin assignments, then locking down the I/O placement so that it reflects the locations chosen by the tools. I/O in the tutorial schematics must be assigned pin locations so that the Calc design can function in the Xilinx demonstration boards. Because the design is fairly simple, these pin assignments do not adversely affect the ability of PPR or APR to place and route the design completely.

Pin locations are specified by attaching a LOC property to the net attached to the pad. LOC properties should not be attached directly to I/O pads. Properties are not associated with nets, only with vertices on nets. When attaching properties, if the center of a net is selected, the entire net segment appears highlighted, indicating that two net vertices are selected, one at each end of the net segment. If a property is then attached to the net, it appears twice when placed, indicating it has been attached to both net vertices associated with the segment. While this is not illegal, it does clutter the schematic. To prevent this, select only one vertex before attaching properties. To select a net vertex, position the cursor *exactly* above the point where the net attaches to the pin, or above the point where the net bends. Otherwise, an entire net segment is selected. This operation is simplified because default pin locations are included with the I/O pins; for example, the “PXX” on the opad symbols. You can modify the existing property, rather than adding a new one.

Modify the LOC property on the pad associated with the LED0 signal on the Calc schematic as follows:

1. Position the mouse over the “PXX” text to the right of the pad attached to net LED0; this is the default location property attached to the net. Refer to the following figure.
2. Without moving the mouse, press Shift-F7. A dialog box appears.
3. If targeting an XC3000 family device, modify the “PXX” text to read **P29**. For XC4000 family devices, type **P60.**
4. Click on **OK** or press return to execute the command.

For simplicity, the other pin locations for the Calc design have been placed in a data file known as a constraint file, which is described in a later section. You can leave the other location values undefined. Valid pin locations vary depending on the package. PLCC package pins are designated with a P followed by the pin number, such as P17. Pin Grid Array (PGA) package pins use alphanumerics such as A12. *The Programmable Logic Data Book* lists the pinouts of each FPGA for each package that Xilinx supplies.

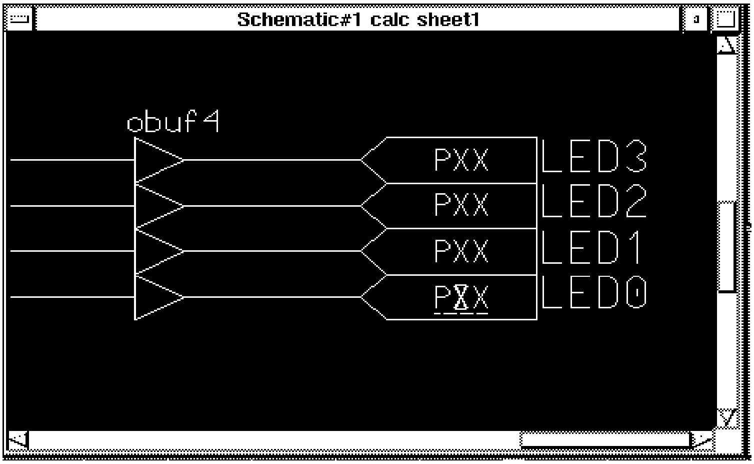


Figure 11-42 Assigning a Location to an Output Net

Designating FAST Pads

Output slew rate can be modified by assigning a FAST attribute to the output buffer, as shown in the following figure. The default slew rate is SLOW. “Fast” pads have different timing specifications and draw more current than “slow” (slew-rate-limited) pads. Slow pads are used by default. In XC4000A devices, the MEDFAST and MEDSLOW properties can also be placed on output buffers, to indicate that XC4000A devices have four different selectable output slew rates, compared to the two available in the standard XC4000 family. See *The Programmable Logic Data Book* for timing specifications for the various slew rate modes.

Add a FAST attribute to the led output display drivers attached to the STACK(3:0) bus as follows:

1. Press Shift-F8 to display the entire Calc schematic.
2. Click the left mouse button on the obuf4 symbol attached to the stack (3:0) bus.
3. Select **Right Mouse Button → Properties → Add**. A dialog box appears.

4. In the both the Property Name and Property value fields type the word **`fast`**.
5. Press return or select **OK** to execute the command.
6. Use the left mouse button to place the text near the obuf4 symbol, as shown in the following figure. Since the property is attached to the obuf4 symbol, it affects all four of the LED outputs.

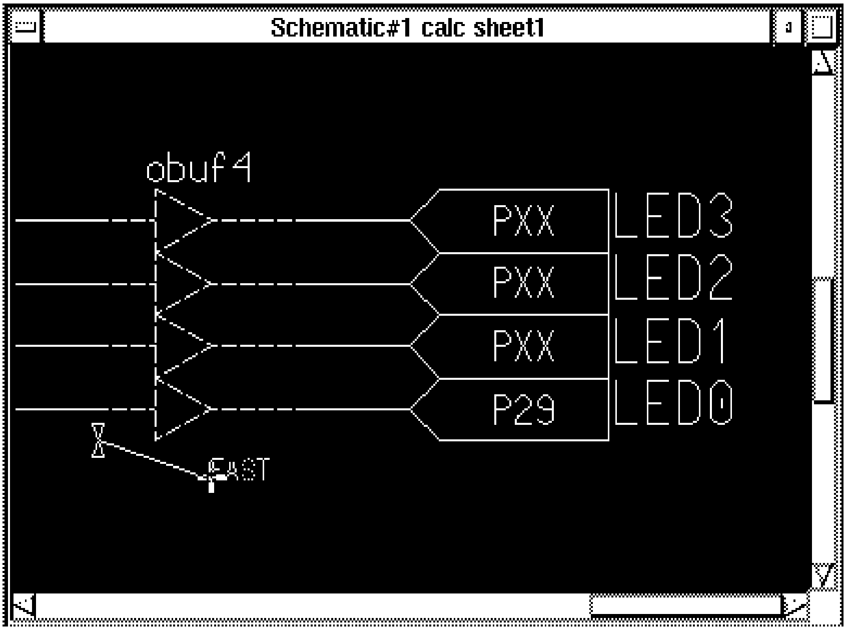


Figure 11-43 Designating a FAST Pad

Using the I/O Flip-Flops

Xilinx XC3000 family devices, XC4000 devices, and XC4000A devices have two flip-flops in each Input Output Block (IOB). Each pad has an associated input flip-flop and output flip-flop. You can also configure input flip-flops as latches and output flip-flops as 3-state. You access these elements using the library components IFD, ILD, OFD, and OFDT, as well as other higher-level macros that contain these components. For more information on these library elements, consult the *XACT Libraries Guide.*

IOB flip-flops are used whenever possible to free up internal CLB resources. IOB flip-flops are used to register the switch inputs. As shown in the figure below, the IFD8 macro attached to the input bus SW<7:0> in the lower-left area of the schematic has an underlying schematic that consists of eight IFD Xilinx primitives. If a similar data register, such as RD8, had been used instead, the flip-flops in the IOBs would be wasted and would occupy valuable CLB resources.

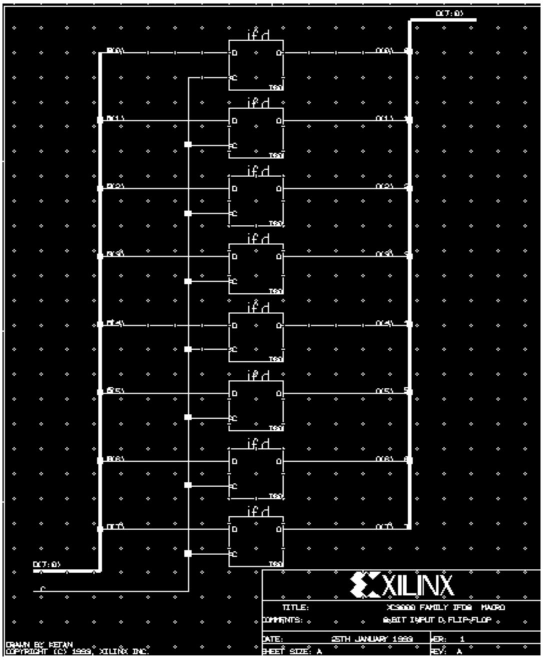

Q<7:0>
ifd
D
Q
C
IFD
D<7:0>
C
XILINX
TITLE: XC4000 FAMILY IFD8 MACRO
COMMENTS: 8-BIT INPUT D FLIP-FLOP
DATE: 25TH JANUARY 1993
VER: 1
SHEET SIZE: A
REV: A
DRAWN BY KETAN
COPYRIGHT (C) 1993, XILINX INC.

Figure 11-44 Underlying IFD8 Schematic Using Input Flip-Flops

Saving the Calc Schematic

Before continuing, check and save the changes made to Calc, as shown earlier in this tutorial.

Optimizing the Design for the XC4000 Family

At this point in the tutorial, you have created or edited the following four schematic files: calc, alu, andblk2, and orblk2. The design, at this point, is suitable for use in an XC4000 family device. Other than changing the references in the design to the XC4000 library, you do not need to make any changes to adapt it to the XC4000 family. However, XC4000 family devices have several advanced features that are not used by this device-independent design. Two of these advanced features are the on-chip memory built into the XC4000 CLB and wide-edge decoders.

Device-Independent Stack Implementation

The device-independent stack is implemented as a set of registers and muxes. This implementation can be used for any Xilinx device.

View the stack as implemented in the device-independent design as follows:

1. Deselect everything on the schematic.
2. Use the left mouse button to select STACK.
3. Open the schematic underneath STACK. The schematic for STACK appears on the screen, as shown in the following figure. Since there are sixteen flip-flops in the block, the most efficient implementation of this logic uses a minimum of eight CLBs.
4. Close the STACK schematic and return to the Calc schematic.

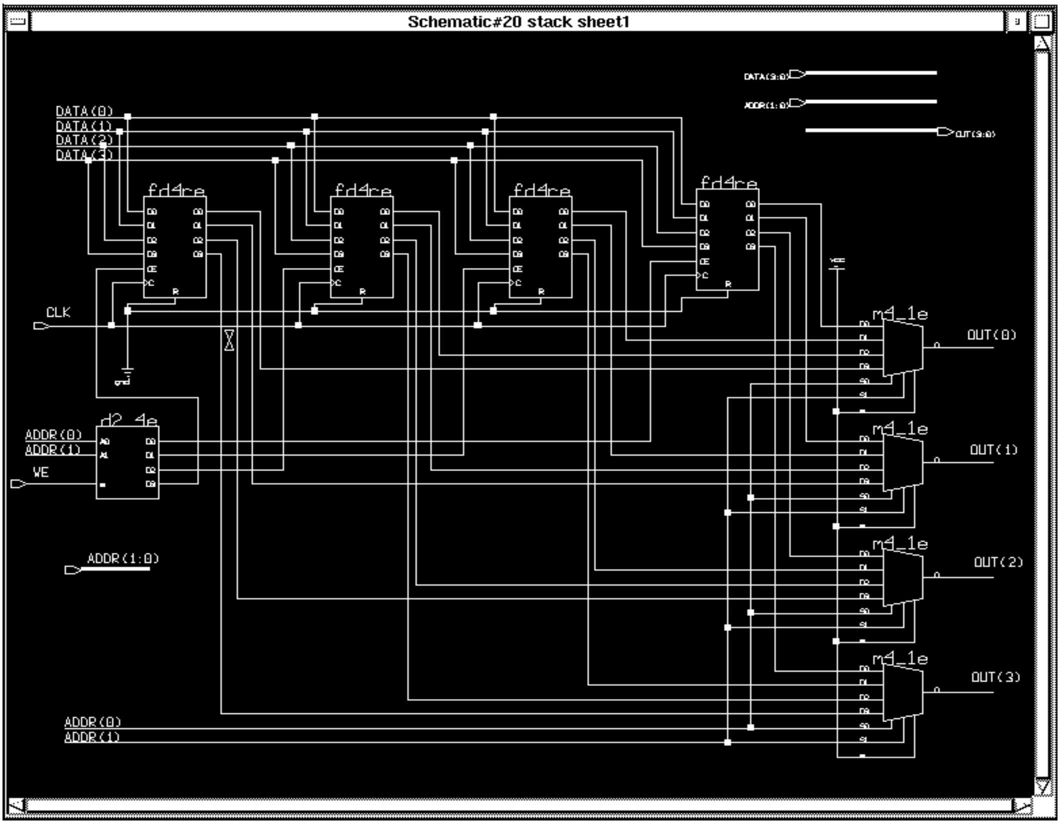


Figure 11-45 Device-Independent Stack Implementation

RAM Stack Implementation (XC4000 Family Only)

The RAM stack is implemented using a single element from the XC4000 library. Although the stack is 4x4, RAM and ROM are only available in 16x1 or 32x1 increments, so only one fourth of the memory addresses are used. A stack four times as deep could be implemented while still using only two CLBs. An equivalent flip-flop implementation would require 64 flip-flops or 32 CLBs.

Using the static memory feature of the XC4000 family CLB, you can reduce the stack from eight CLBs to two CLBs as follows:

1. Make sure the STACK symbol is selected in the Calc schematic.
2. Replace the device independent stack with the RAM stack by selecting **Right Mouse Button → Replace → Other**.
3. Use the dialog box that appears to replace STACK with $xilinx_tutorial/calc_da/stack_4k.
4. The STACK_4K symbol does not have a CLK pin. The RAM stack implementation does not require a clock. If desired, you can select the unconnected clock segments and remove them using the delete command. Make sure nothing else is selected when this is done.
5. Unselect everything, then select the STACK_4K symbol and open the underlying schematic. The schematic appears as shown in the following figure.
6. Close STACK_4K and return to the Calc schematic.
7. Check and save the changes to the Calc schematic.

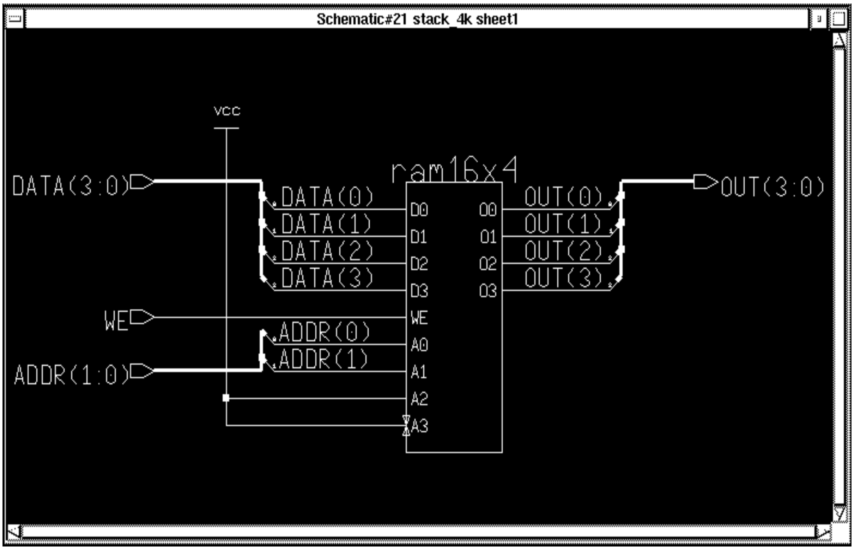


Figure 11-46 XC4000 Family RAM Stack Implementation

Device-Independent State Machine

The device-independent control logic is implemented as a set of flip-flops forming a three-state one-hot state machine and some additional logic gates. This implementation can be used in any Xilinx device. View the state machine and control logic as implemented in the device-independent design as follows:

1. Unselect everything by pressing F2 key.
2. Select CONTROL using the left mouse button.
3. Open the schematic for CONTROL. The CONTROL schematic appears on the screen.
4. Making sure nothing else is selected, select STATMACH.
5. Open the schematic for STATMACH. The schematic for STATMACH appears on the screen, as shown in the following figure. Note that the signals RST, SEL_OP, and UP_DN are all ANDs of opcode inputs and the EXC signal. This is important in a later step.
6. Close the STATMACH schematic and return to the CONTROL schematic.

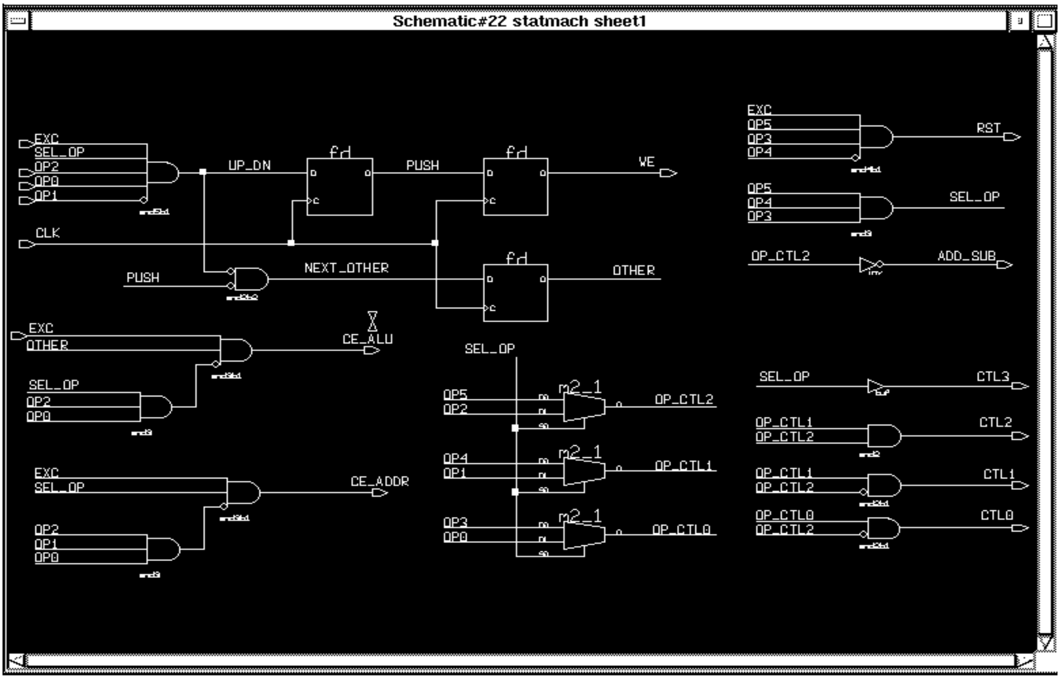


Figure 11-47 Device-Independent State Machine Implementation

Wide-Edge Decoders (XC4000 Family Only)

XC4000 family parts have wide-edge decoder lines on each edge of the device. Each decoder line can be driven by many edge decoders. Some edge decoders have dedicated connections to pads, while others can be driven from internal logic. You can view this as a wire to which many wired-and outputs can be connected.

For this design, all opcode pins are on the left side of the device in the 4003A and 4003 parts. EXC is an internal net. You can implement four of the five signals in the STATMACH schematic using wide-edge decoders. Do not use a wide-edge decoder to implement an AND gate unless there are at least ten inputs to the gate, since a CLB implementation is faster and routes more easily for an AND function with nine or fewer inputs. In a dense design, however, using edge decoders can be advantageous because it frees up CLB resources.

When you use a wide-edge decoder, make sure that the pads associated with the inputs are all on the same edge of the device.

Although the decoders in this design do not have ten inputs, the control logic in the STATMACH block is implemented using wide-edge decoders as an example. See *The Programmable Logic Data Book* for additional information on wide-edge decoders in XC4000 family devices.

Replace the STATMACH symbol with the STATE_4K symbol as follows:

1. Make sure the STATMACH symbol is selected.
2. Replace the STATMACH component with $xilinx_tutorial/calc_da/state_4k using the **`Right Mouse Button`** → **`Replace`** → **`Other`** command.
3. Open the underlying schematic for STATE_4K using **`File`** → **`Open Down`** from the menu bar. The STATE_4K schematic appears on the screen, as shown in the following figure.
4. The decode logic is implemented using the elements DECODE4, DECODE8, and so on. A pull-up must be placed on the output. Tie unused inputs to a VCC symbol.
5. Close STATE_4K and return to the CONTROL schematic.
6. Check and save the changes to the CONTROL schematic. If errors occur during the check, fix them and then re-check the schematic.
7. Close the CONTROL schematic and return to the top-level Calc schematic.

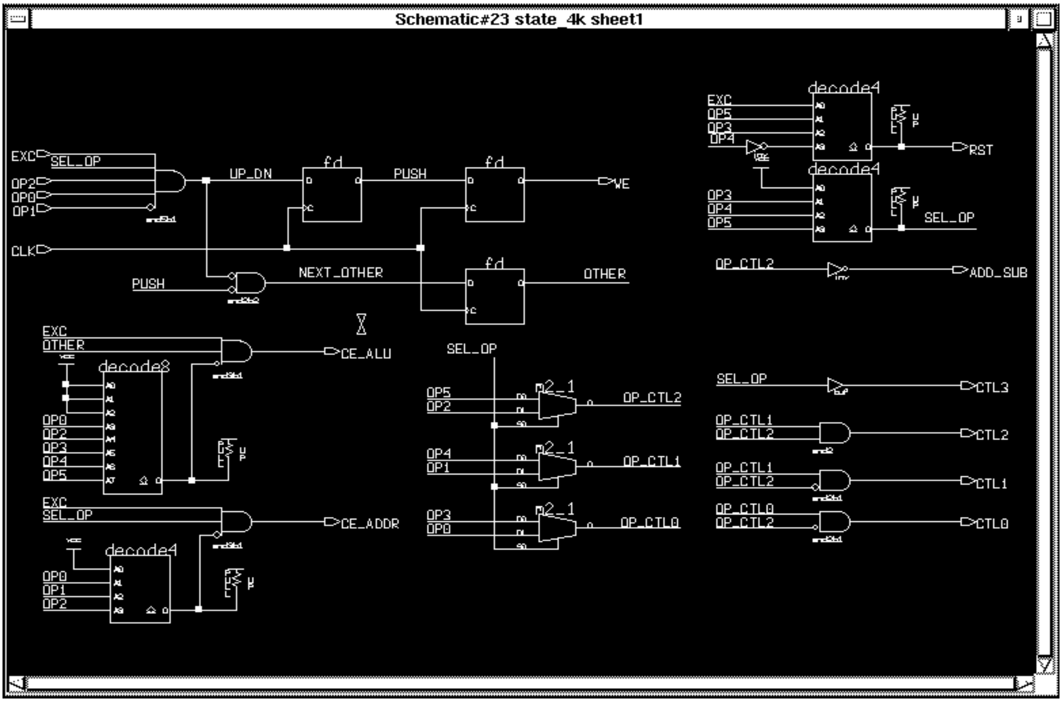


Figure 11-48 XC4000 State Machine with Wide-Edge Decoders

Configuring XMake using XDM

The XACT Design Manager (XDM) is a program similar to PLD_DMGR. It provides you with an interface to run the various programs necessary to implement a design. The XMake program can be run from within both PLD_DMGR and XDM. XMake automatically converts your design to a bitstream that you can use to program a part. You can use XDM to configure and run the individual programs that are run by XMake. You cannot perform this task in PLD_DMGR. See the “Command Summaries” section at the end of this tutorial for a list of XMake programs.

Using a Constraints File

Using a constraints file, you can supply constraints information in a textual form rather than putting it on a schematic. Sometimes this method is more efficient than putting constraints on a schematic. An example of a constraints file is shown in the figure below. The figure shows the constraints file, calc_4k.cst, that is supplied with this tutorial. The constraints file syntax is slightly different for XC3000, XC3000A, and XC4000 family designs.

If there is a constraints file in the design directory with the same name as the top-level schematic, it is used by PPR. If you do not want the constraints file applied, you must disable this function in the Profile Options menu. APR never automatically reads constraints files; it must be explicitly instructed to do so. Constraints file syntax is different for PPR and APR. Also see *XACT Reference Guide*.

```
# This is a schematic constraints file for use with the
#   Calc Tutorial Design. It maps the I/O to the correct
#   pins for 4K family parts on the Xilinx demo boards.
# When using this constraints file, you must tell both
#   XNFPrep and PPR to ignore pad locations on the
#   schematic, as the locations on the schematic are for
#   3K family parts.
# Set the ignore_locs options for XNFPrep and PPR to a
#   value of “IO”.
place instance   A:    P49;
place instance   B:    P48;
place instance   C:    P47;
place instance   D:    P46;
place instance   E:    P45;
place instance   F:    P50;
place instance   G:    P51;
place instance   SW<7>:   P19;
place instance   SW<6>:   P20;
place instance   SW<5>:   P23;
place instance   SW<4>:   P24;
place instance   SW<3>:   P25;
place instance   SW<2>:   P26;
place instance   SW<1>:   P27;
place instance   SW<0>:   P28;
place instance   LED3:  P57;
place instance   LED2:  P58;
place instance   LED1:  P59;
place instance   LED0:  P60;
```

Figure 11-49 calc_4K .cst File

Since you only specified a pin location for one of the many inputs and outputs on the Calc schematic, you must use a constraints file to find the rest. You can do this in XDM by specifying an option for the place and route tool as follows:

1. Quit PLD_DA and enter PLD_DMGR.
2. Execute XDM from PLD_DMGR by double-clicking on the PLD_XDM icon. The XACT Design Manager appears.
3. Verify that the directory in XDM is set to $xilinx_tutorial / calc_da directory. The directory is displayed in the lower left corner of the XDM screen. $xilinx_tutorial is not displayed in XDM, but the value assigned to $xilinx_tutorial is displayed.If the directory is incorrect, click on **Directory:** and change the directory in the dialog box.
4. In the lower left corner, verify that the family is set correctly for the device being targeted. If not, click on **Family**: and choose the correct family from the dialog box that appears.The part type can be set to InDesign, or to any part, since its value is not used by the programs in this tutorial.
5. Select **Profile** → **Options** from the menu bar. A list of available programs appears.
6. If using an XC3000A or XC4000 family device, select **PPR** → **helpall** → **cstfile**. Choose **calc_3ka.cst** for XC3000A designs and **calc_4k.cst** for any XC4000 family design. If using an XC3000 device, select **APR** → **-C.** Select **calc_3k.cst** to supply pad locations for the XC3020PC68 device in the Xilinx demonstration boards.
7. Select **Done** → **Done** to return to the XDM executive menu.
8. To save all of the changes that you just made to the XDM and XMake defaults, pull down the **Profile** menu and select **Saveprofile** → **Yes**.
9. Select **Quit** from the menu, then select **Yes** from the dialog box that appears.
10. Return to the PLD_DMGR window.

Using .PRO Files

The Saveprofile command saves changes to the xdm.pro file in your design directory. There are several .pro files associated with the Xilinx software; they are used to set default options. The xdm.pro file sets options that are used by XMake, whether or not it is called from XDM, and by any other tools called from XDM. If you do not use XDM and run programs from the UNIX command line, you must enter the parameters at the command line or in a batch file.

Some default profiles are supplied with the Xilinx software, and some are created the first time you run a program. Default files are located in the $XACT/data directory. If the appropriate .pro file exists in the design directory, it is used; if not, the software uses the one from the $XACT/data directory. It is common practice to keep .pro files in the design directory to customize your environment. By doing this, you can avoid changing the program options each time you run the software. User .pro files are saved to the design directory, not to the $XACT/data directory. The files there are retained as templates. They are overwritten each time the software is updated or reinstalled.

Using PLD_Men2XNF8

PLD_Men2XNF8 is a Design Manager icon. Running PLD_Men2XNF8 is always the first step in simulating or implementing a design. This icon allows you to create a Xilinx netlist from your Mentor Graphics design. You must run PLD_Men2XNF8 again if you make changes to the schematic.

When you run PLD_Men2XNF8 a dialog box with the following options appears:

- Design Object: enter the name of the Design Object you want to process.
- Part Type: Type the part type in this field: 3020PC68, 3020APC68, 4003PC84, or 4003APC84.
- Run Memgen only?

  If selected, Men2XNF8 does not generate a netlist for the selected design object. Instead, it searches the design directory and runs the MemGen program on .mem RAM or ROM description files.

- Verbose Output?

  If chosen, PLD_Men2XNF8 provides additional output to the screen.

- Help?

  If the Help option is chosen, a short help listing is produced by the Men2XNF8 script.

Perform the following steps for the Calc design:

1. Double-click on the PLD_Men2XNF8 icon.
2. In the Design Object field type **`$xilinx_tutorial/calc_da/calc.`**
3. Type the correct Part Type in the Part Type field.
4. Select **`Yes`** for Verbose Output.
5. Leave the other options set at the defaults. Press return or choose **`OK`**.
6. PLD_Men2XNF8 is run and produces an XNF netlist. “Done” appears at the bottom of the window that was created by PLD_Men2XNF8 when the program is finished. Dismiss the window by typing CTRL-C in it.

Examining PLD_Men2XNF8 Output Files

In addition to the XNF netlist, PLD_Men2XNF8 also creates the men2xnf8.log file. This file contains a transcript of the outputs of the programs run by PLD_Men2XNF8. If the program fails before creating an XNF netlist, the error is logged in this file. Other files that contain useful information are referenced in men2xnf8.log.

Examine the men2xnf8.log file for the Calc design as follows:

1. Select the navigator window.
2. Choose **`Right Mouse Button → Update Window.`**This updates the navigator window to display the new files created by PLD_Men2XNF8.

3. Select the men2xnf8.log file and choose **`Right Mouse Button` → `Open` → `Read-Only Editor.`** A text window appears containing the log file. When you are done viewing it, close the window.

Using PLD_XMake

PLD_XMake is a Design Manager icon that allows you to automatically implement your design.

When you run PLD_XMake a dialog box with the following options appears:

- Design Object: enter the name of the Design Object you want to process.
- Override Part Type?

  When PLD_Men2XNF8 is run, it inserts the selected part type into the netlist that is read by PLD_XMake. This option allows you to override the part type without having to run PLD_Men2XNF8 again. If you select Yes, a field appears, and you can enter the new part type.
- Verbose Output?

  If chosen, PLD_XMake provides additional output to the screen.
- Rerun all Steps?

  If interrupted and then run again, PLD_XMake can determine where in the place and route process it was stopped, and then resume processing at this point. If this option is selected, PLD_XMake restarts the place and route process using the netlist from PLD_Men2XNF8. This option is useful if, for example, a change is made to a constraints file, and you want to see how this affects the design. In this case, you would select Yes for this option.
- Use Guide File?

  A placed and routed design from a prior iteration of PLD_XMake can be used to guide the place and route of another design iteration after small modifications to the design have been made. When you select this option, the place and route time is usually significantly improved and the timing of the unchanged parts of the design remain more stable over multiple place and route runs.

- Perform X-BLOX Optimization?

  Use this option to force the execution of the X-BLOX program on a design, even on one that does not use X-BLOX symbols, in order to take advantage of the optimization X-BLOX performs.

- Generate MAK File Only?

  Use this option to create a MAK file only, without running the commands in the MAK file to implement the design. Select this option when you want to create a custom MAK file, forcing XMake to generate a script that you can edit.

- Output to Screen?

  If selected, this option directs the output of all programs run by PLD_XMake to the screen.

- Mapping Strategy

  It is sometimes preferable to map sections of a design separately before merging the different sections together. This option gives you control over the mapping strategy. Refer to the sections on PPR, XNFMap, and XNFMerge in the *XACT Reference Guide* for more information on mapping strategies.

- Target

  By default, PLD_XMake processes a design all the way to a configuration bitstream BIT file, unless you specify a different target. XMake will stop after creating the specified target file. After stopping, PLD_XMake can be run again with a different target, and executes the commands that were not run in the previous iteration.

Perform the following steps for the Calc design:

1. In the Design Object field type **`$xilinx_tutorial/calc_da/calc.`**
2. Leave the rest of the options set to the defaults. Press return or choose **OK**. PLD_XMake spawns a shell and runs XMake in it with the selected options. Fix any errors, as described in the next section. If there are no errors, a message appears indicating that the calc.bit file was successfully created.

Examining PLD_XMake Output Files

In addition to the routed LCA file and the bitstream BIT file, PLD_XMake generates three useful output files as described below:

- OUT File

  For the Calc design, the file calc.out contains a copy of all text that is echoed to the screen. You should always review the OUT file after running XMake, even if you did not see any warnings or error messages while the design was being translated. If there are any error messages, it is easier to resolve problems at this point in the design process rather than in later steps. The OUT file lists every program run by XMake.

- PRP File

  For the Calc design, calc.prp is the design rule check report file generated by the XNFPrep program. If XNFPrep finds any errors or warnings, the OUT file directs you to examine this file. Other than an explanation of errors and warnings, the PRP file contains a detailed list of all logic trimmed by XNFPrep and the reason why it was unnecessary. This file can be a very useful debugging tool.

- RPT File

  For the Calc design, calc.rpt is a report of the results of the placement and routing. It is generated by the place and route software, either PPR or APR, depending on the family. Check the RPT file to make sure there were no unrouted pins or nets.

Examine OUT, PRP, and RPT files for the Calc design as follows:

1. Select the navigator window.
2. Choose **Right Mouse Button → Update Window.**This updates the navigator window to display the new files that were created by PLD_XMake.
3. Select the calc.out file in the navigator and choose **Right Mouse Button → Open → Read-Only Editor**. A text window appears containing the .out file. When you are done viewing it, close the window.
4. Repeat step three for the .prp and .rpt files.

Checking for Warnings in the OUT and PRP Files

You should expect to see some warning messages in the calc.out and calc.prp files, but no errors. Errors are problems with the design that cause PLD_XMake to terminate. Warnings tell you that there are unusual aspects to your design. You can choose to correct the reported problems. A partial listing of calc.out for the XC3020APC68 is shown in the following figure.

In the calc.out file, PPR reports a warning that the Calc design contains a combinational loop. This loop is intentional. The OSC_3K schematic uses this logic loop with an external RC circuit to generate the clock. You can ignore this warning if you targeted the Calc design to an XC3000 family part.

Correct any other errors or warnings before continuing. To find the source of unexpected errors, reload the design in Design Architect and compare the schematics to the schematics shown in the figures in this chapter. After correcting the errors, save the changes, run the Check program, and run PLD_XMake again.

```
XMAKE: Begin command ‘xnfmerge -D xnf -D. -P3020APC68-6 xnf/
calc.xnf calc.xff’.
Netlist written to file calc.xff

********************************************************
XMAKE: Begin command ‘xnfprep calc.xff calc.xtf
parttype=3020APC68-6’.

xnfprep: running design rule checker ...
xnfprep: checking XACT-Performance specifications ...
xnfprep: trimming unnecessary and redundant logic...
xnfprep: running design rule checker on trimmed design...
xnfprep: reverifying XACT-Performance specifications in trimmed
design ...

XNFPREP SUMMARY
 ---------------
0 Errors found
2 Warnings found
16 Unnecessary inverters and buffers removed
32 Unnecessary or disabled symbols removed
27 Sourceless or loadless signals removed
Refer to the calc.prp file for details.

********************************************************
XMAKE: Begin command ‘xnfmap -P3020APC68-6 calc.xtf calc.map’.

DESIGN SUMMARY:
```

```
Part type=3020APC68-6
50 of 64 CLBs used
22 of 58 I/O pins used
0 of 6 internal IOBs used
0 of 16 internal three-state signals used (0 TBUFS used)
28 CLB flip-flops used
Total number of WARNINGS generated during mapping = 0.
Total number of ERRORS generated during mapping = 0.

Writing design file calc.map...
Writing guide file calc.pgf...

********************************************************
XMAKE: Begin command ‘ppr calc.map parttype=3020APC68-6’.

ppr: Routing signals...
ppr: Generating .LCA File...

ppr: Generating .BID Back Annotation File...
ppr: Making Report File...
Wrote report on the result without delay optimization to
calc.rpf.
ppr: Routing signals...

*** PPR: WARNING:
This design has 1 purely combinational loop. Such loops should
be avoided. If at all possible, please modify the design to
eliminate all unclocked feedback paths.

A loop of 1 source-to-load connections: FG FG_OSC_3K/Q (Net
OSC_3K/Q) to first gate again.

No more unroutes
Therefore deleting result with 1 unroute
Begin work on a 65.5ns path with 9 pins.
Design has 0 unroutes.

--------------------------------------------------------
Timing analysis summary
--------------------------------------------------------

Deadline Actual(*) label: [qualifier]
-------- --------- -----------------
pad to pad <auto> 38.4ns <default>
-----------------------
Selector net: Default
-----------------------

clock to setup <auto> 59.9ns
pad to setup <auto> 12.0ns
clock to pad <auto> 42.6ns

(*) Note: please run xdelay to confirm the actual path delays
```

```
computed by PPR.

# of unrouted connections: 0.
ppr: Making Report File...

*********************************************************
XMAKE: Begin command ‘xdelay -D -W calc.lca’.

xdelay: writing calc.lca with delay information...

*********************************************************

XMAKE: Begin command ‘makebits -R2 -S0 calc.lca’.

Timing nets....
```

Figure 11-50 XMake Partial calc.out File

The OUT file for the XC3020APC68 design shows that XNFPrep issued two warnings and no errors. The OUT file directs you to the PRP file for explanations about the warnings.

When you examine the calc.prp file, you may see warnings similar to those shown in the following figure. The PRP file includes an explanation of these warnings. The warnings listed in calc.prp are for your information only and do not require any further action.

```
---------------
XNFPREP: WARNING 4037:

These inverters could not be absorbed and each will be
implemented in a single function generator. This will introduce
additional delay and use resources inefficiently. (Note that
some of the symbols listed below may have been reduced to
inverters by earlier trimming.)

Inverter Name = DEBOUNCE/$1I27
Output Signal = DEBOUNCE/$1N14

Inverter Name = ALU/$1I195
Output Signal = $1N403

Inverter Name = CONTROL/$1I31/Q0/$1I30/$1I8
Output Signal = CONTROL/$1I31/Q0/MD

---------------

XNFPREP: WARNING 4082:

Double pull-ups were found on TBUF-driven longlines and/or edge
decoder longlines. Requiring two pull-ups would prevent half-
length longlines from being used, and design placement and
resource utilization would be adversely affected.
```

```
One pull-up is being removed from each of these longlines. PPR
will activate both pull-ups if the signal is routed using a
complete longline.

 ---------------

XNFPREP: WARNING 4613:

Each of the following signals drives more than 20 inputs.

Signal Name
-----------
ADDR0
ALU3
ALU2
ALU1
ALU0

Although this is valid there is a possibility that the routing
delays of these signals will be unacceptable. If these signals
are critical to the timing of your design, you may consider
replicating the logic used to generate the signals so that the
loading (and hence the subsequent routing delay) is reduced.
```

Figure 11-51 XNFPrep Warning Messages

Checking the RPT File

The report file contains detailed information on your routed design. Useful information in the RPT file includes the pinout description and detailed timing information for PPR. A portion of the report file for the XC3020APC68 Calc design is shown in the following figure:

```
PPR RESULTS FOR DESIGN CALC Xilinx, Inc.

Design Statistics and Device Utilization

----------------------------------------
Design Summary from XNF File
Number of Logic Symbols: 0
Number of Signals: 132
Number of Pins: 487

Equivalent “Gate Array” Gates: 198

Partitioned Design Utilization Using Part 3020APC68-6

                            No.Used Max Available % Used

--------------------------- ------- ------------ ------
Occupied CLBs                     50       64         78%
Packed CLBs                       35       64         54%
---------------------------- ------- ------------ ------
```

```
Package Pins:                     22       64          34%
F and G Function Generators:      71       128         55%
H Function Generators:            0        64          0%
CLB Flip Flops:                   35       256         13%
Memory Write Controls:            0        64          0%
3-State Buffers:                  0        192         0%
3-State Longlines:                0        32          0%
Edge Decode Inputs:               0        96          0%
Edge Decode Longlines:            0        32          0%

Routing Summary
Number of unrouted connections: 0

Cpu Times
Reading Input: 00:00:15
Partition: 00:00:17
Placement: 00:02:36
Routing: 00:01:50

Chip Pinout Description
-----------------------

This chapter describes where your design’s pins were placed in
terms of the package pins. This first list is sorted by package
pin location. The second list presents the same data sorted by
your design’s pin names.

Package Pin Location Pin Name
-------------------- -----------------
P11 :                  EXC_P
P12 :                  OSC_3K/CQL
P13 :                  SW7/SW6_P
P14 :                  OSC_3K/CQ
.
.
P55 :                  7SEG/G_P
P56 :                  7SEG/D_P

This list describes where your design’s pins are in terms of the
package pins; it is sorted by your design’s pin name. The list
presented above has the same data sorted by package pin
location.

Package Pin Location Pin Name
-------------------- -----------------
P11 :                  EXC_P
P29 :                  LED/LED0_P
P30 :                  LED/LED1_P
P31 :                  LED/LED2_P
.
.
```

```
P53 :                   7SEG/F_P
P55 :                   7SEG/G_P
```

Figure 11-52 Partial calc.rpt File

Examining Routed Designs with XDE

Note: The XACT Design Editor (XDE) is not available with the Base Development System. You can omit this section without affecting the outcome of the tutorial. If you skip this section, continue with the next section, “Verifying the Design Using a Demonstration Board.”

At this point in the tutorial, the design process is complete. You can take a graphic look at your placed and routed design using the EditLCA program, which is included in the XACT Design Editor (XDE). You can access XDE from XDM or from the UNIX command line.

XDE and EditLCA provide several useful functions, such as:

- Manual editing of a routed design
- Probe insertion during in-circuit verification
- Static timing analysis

For more information on XDE, see the *XACT User Guide*.

Entering the Design Editor

1. Double-click on the PLD_XDM icon in PLD_DMGR to execute XDM.
2. Pull down the PlaceRoute menu and select **XDE**. An options menu appears. Use the default options.
3. Select **Done**. The XDE Executive screen appears. In the bottom left corner of the screen, Mode: Safe, appears. This mode prevents you from making any changes to the LCA file that changes the functionality of the design. To make changes to a design using EditLCA, you must change to Expert mode before loading the design.
4. Click on Mode with the left mouse button. A menu with the choices, Safe or Expert, appears.
5. Click on Cancel to remain in Safe mode.

6. To load the design into the editor, select **`Design`**. A pull-down menu appears.

7. To choose the input LCA file, select **`Design`**. A menu of available LCA files appears. In this case, there is only one LCA file in the directory, calc.lca, the routed file generated by XMake.

8. Select **`CALC.LCA`**. The name of the design file appears in the status area above the command line. When you select the design, the part type is automatically set to the part type in the selected LCA file.

9. To enter the Design Editor, pull down the Programs menu and select **`EditLCA`**. The design begins loading into the editor. Note the status line above the command line to watch the progress of XDE. The following messages appear.

   ```
   Loading die/package file...
   Loading design file...
   Building pip drawing information...
   Drawing screen...
   Timing nets...
   ```

10. The editor appears, as shown in the following figure. The editor is a graphic representation of the LCA. Pan around the device by holding down the left mouse button and moving the mouse. A world view of the LCA appears in the lower right corner of the screen, and a red box shows the current location on the device.

11. Two types of blocks are shown in the editor. The IOBs appear around the periphery of the device, and the CLBs appear in the middle. Pan around the screen to see how the design was placed and routed in the device. Used blocks are highlighted, and the signal nets connecting them are shown as highlighted traces.

12. Looking at the routed design, observe how the global clock was laid out in the device and how the pin location constraints were carried through from the schematic level to the routed design.

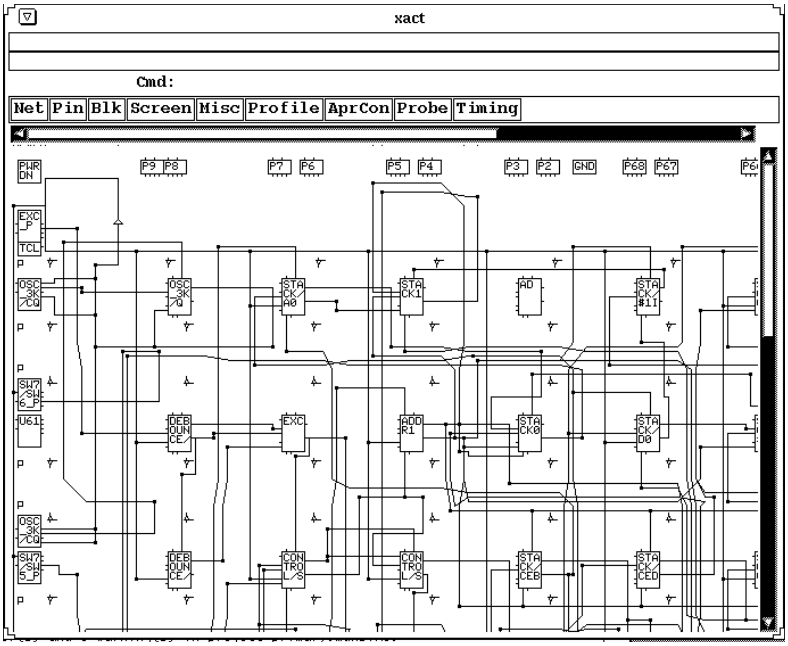


Figure 11-53 Portion of XDE EditLCA Screen, XC3020APC68

Finding a Block

You can locate the global clock buffers as follows:

1. Pull down the Screen menu and select **Find**.
2. Now enter the name of the global clock buffer, **gclk** ↵ for an XC3000 family design or **bufgs_tl** ↵ for an XC4000 family design, to select the buffer in the top left corner. The cursor moves to the top left corner of the LCA editor. This is the location of the global clock buffer on XC3000 devices and one of the global secondary clock buffers on XC4000 devices. This buffer may be the one that PPR used to generate the clock signal in your XC4000 design.
3. Select **Done** to exit the Find command.

Highlighting a Net

The detailed operation of XDE is beyond the scope of this tutorial, but there are a few commands that are especially useful for examining this design. The Net Hilight command is used to highlight the path of a particular net to make the net easier to trace across the device.

1. Pull down the Net menu and select **Hilight**. A menu of colors appears.
2. Select a highlight color to use.
3. Type **clk** ↵. The clock net appears in the selected color.
4. Select **Done** from the top of the screen to finish the Net Hilight command.

Using Command Line Entry

XDE commands can also be entered directly from the keyboard instead of using the pull-down menus.

1. To remove the highlighting from the global clock net, type **unhilight clk** ↵ on the command line.
2. Leave EditLCA by typing **quit** ↵. The XDE Executive menu returns.

Running the Design Rule Checker

XDE has a design rule check program, DRC, which ensures that an LCA design is valid. If you make any edits in EditLCA, you should run the DRC program to make sure you have not introduced any invalid connections. You can also run DRC from inside the EditLCA program.

To run DRC from XDE, perform the following:

1. Pull down the Programs menu and select **DRC**. A menu of options appears.
2. None of these options are necessary for this design; select **Done**. The screen switches to text mode and messages describing what the DRC program is checking appear. No errors or warnings should occur.
3. Press any key to return to the XDE Executive Screen.

4. Exit XDE by typing **quit** ↵.
5. Press any key to return to XDM.

Verifying the Design Using a Demonstration Board

Now that you have completed your design and run a design rule check, you are ready to download it to an FPGA. The design bitstream was created by PLD_XMake.

There are three Xilinx demonstration boards that are commonly used. Which board you have depends on what software you purchased and when you bought it. Your tutorial design should be targeted to a device on the board that you have available. The three boards are as follows:

- The XC3000 / XC4000 board has both an XC3000 family socket and an XC4000 family socket, containing an XC3020APC68 and an XC4003APC84. Use only one of the two parts for this tutorial.
- The XC3000 demonstration board contains a single XC3020PC68.
- The XC4000 board contains a single 4003PC84 or 4003APC84.

To load the configuration bitstream to the demonstration board, you need a Xilinx hardware cable. Xilinx makes two hardware cables, the XChecker cable and the Download cable. Either cable works with any of the Xilinx demonstration boards.

Connecting the Cable for Download

Before initiating the downloading of the design into the LCA, the demonstration board must be correctly hooked up to your workstation.

There are several control and power pins that must be connected between the board and the cable. The bundles of leads supplied with the cables are labeled to make connecting the board to the cable a simple procedure. Additionally, a pair of power and ground pins must be connected to a regulated 5 volt power supply to supply power to the board and cable.

Connect one end of the cable to your demonstration board as described in the appropriate table below. For the XC3000 / XC4000 demonstration board, use the leftmost column of pins, which is missing the pin in the third position.

Table 11-3 Demonstration Board Cable Connections For XC3000

XC3000 Board	Cable Label	XC3000/ XC4000 Board	Cable Label
J1-1	V_{CC}	J1-1	V_{CC}
J1-2	Gnd	J1-3	Gnd
J1-3	No Connection	J1-5	No Connection
J1-4	CCLK	J1-7	CCLK
J1-5	D/P	J1-9	D/P
J1-6	DIN	J1-11	DIN

Table 11-4 Demonstration Board Cable Connections For XC4000

XC4000 Board	Cable Label	XC3000/ XC4000 Board	Cable Label
J1-1	V_{CC}	J2-1	V_{CC}
J1-2	Gnd	J2-3	Gnd
J1-3	No Connection	J2-5	No Connection
J1-4	CCLK	J2-7	CCLK
J1-5	D/P	J2-9	D/P
J1-6	DIN	J2-11	DIN
XChecker Cable Only:		**XChecker Cable Only:**	
J1-7	PROG	J2-13	PROG
J1-8	INIT	J2-15	INIT
J1-9	RST	J2-17	RST

The other end of the cable must be plugged into the back of your machine. Attach the cable to your system RS-232 serial port.

The “XC3000 Design Demonstration Board,” “XC4000 Design Demonstration Board,” and “FPGA Demonstration Board” chapters of the *XACT Hardware and Peripherals Guide* discuss in detail the various demonstration boards and how to hook them up. Please refer to this manual for instructions, if necessary, then carefully verify the following:

- Verify that the hardware cable is correctly connected to both your system and the demonstration board. Connections from the cable to the demonstration boards are shown in the two tables shown above.
- Verify that the power supply is connected to the demonstration board *and is turned on*. The power connections for the demonstration boards are shown in the following table.

Table 11-5 Demonstration Board Power Connections

XC3000 Board		XC4000 Board		XC3000/XC4000 Board	
J3-1	+ 5 volts	J2-1	+5 volts	J9-1	+5 volts
J3-2	Gnd	J2-2	Gnd	J9-2	Gnd

XC3000/XC4000 Demonstration Board

Make sure the XC3000/XC4000 FPGA demonstration board is set up for slave mode configuration. The configuration mode for the XC3000 family part is controlled by the SW1 bank of switches. The configuration mode for the XC4000 family part is controlled by the SW2 bank of switches. The switches should be set as shown in the appropriate table below.

Table 11-6 XC3000/XC4000 Board Switch Positions for XC3000

Switch	Label	Setting
SW1-1 (left)	OE / R	Off (unless using battery)
SW1-2	MPE	off
SW1-3	SPE	off
SW1-4	M0	on
SW1-5	M1	on
SW1-6	M2	on
SW1-7	MCLK	off
SW1-8 (right)	COUT	off

Table 11-7 XC3000/XC4000 Board Switch Positions for XC4000

Switch	Label	Setting
SW2-1 (left)	PWR	Off (unless using battery)
SW2-2	MPE	Off
SW2-3	SPE	Off
SW2-4	M0	On
SW2-5	M1	On
SW2-6	M2	On
SW2-7	RST	Off
SW2-8 (right)	No Label	Off

XC4000 Demonstration Board

Make sure the XC4000 demonstration board is set up for slave mode configuration. The configuration mode is controlled by the SW4 bank of switches. The switches should be set as shown in the following table.

Table 11-8 XC4000 Board Switch Positions

Switch	Label	Setting
SW4-7	PWR	Off (unless using battery)
SW4-6	MPE	Off
SW4-5	SPE	Off
SW4-4	M0	Off
SW4-3	M1	Off
SW4-2	M2	Off
SW4-1	RST	On
SW4-0	No Label	Don’t Care

XC3000 Demonstration Board

If you have an XC3000 demonstration board that has been modified for use with a serial PROM, be sure it is not configured for use with an XC1736A or XC1765 Serial Configuration PROM.

Note: If you have the demonstration board as shipped with Xilinx products, there is no serial PROM socket on the board.

Such a modified board contains a four-position DIP switch with a power switch and three switches controlling the programming mode. If this DIP switch is present, make sure that the switches are set for slave mode download. A serial PROM can be present on the board if this DIP switch is installed and set correctly. Information on modifying the demonstration board for use with a serial PROM is available in the “XC3000 Design Demonstration Board” chapter of the *XACT Hardware and Peripherals Guide.*

Downloading the Bitstream

Once the cable is connected to your PC or workstation, you are ready to download the bitstream.

1. Set all of the input switches High. This setting (SW3 switches High on the XC3000 / XC4000 board, SW5 switches High on the XC4000 boards, SW1 switches to “1” on the XC3000 board) selects the No-Op command.

2. In XDM, select the Verify menu.
3. Select **XCHECKER**.

   The XChecker software is used for either hardware cable.
4. Select **-port <name>** and the correct port.
5. Select **Done** and the input file name: **CALC.BIT**.

   Alternatively, you can run XChecker from the system prompt. Type:

   xchecker -port *portname* calc↵

   An example is the following:

   ```
   xchecker -port com2 calc↵
   ```

   Valid port names for a workstation are normally /dev/ttya or /dev/ttyb. Consult your system administrator if these do not work.

   Once you have used XChecker and set the correct port, that information is saved in the file, xchecker.pro, in your design directory, and you do not have to specify it each time.
6. If you are using the Download cable to program an XC4000 family part, press the PROG button. This step is not necessary if you are using the XChecker cable or using the Download cable to program an XC3000 family part.
7. Press the ↵ key.

   If the LCA is successfully programmed, the following message appears:

   ```
   DONE signal went high.
   ```
8. Press any key to return to XDM.

   If the done signal does not go HIGH, check the connections between the cable and the demonstration board, power the board off and on, and try downloading again.

Note: The Download cable has limited functionality when used with XC4000 family parts and can report that Done went HIGH even if you do not press the PROG button. In this case the part is not reprogrammed. Download the bitstream again, this time pressing the

PROG button prior to configuration. Cycling the power off and on before beginning the download has the same effect.

Testing the Design

As described earlier, the Calc design is a four-bit processor with a stack similar to a calculator that uses reverse polish notation. You must supply three types of input: an opcode, data, and an execute command.

Each demonstration board has a row of eight rocker switches that provide input to the design (SW3 on the XC3000 / XC4000 board, SW5 on the XC4000 board, SW1 on the XC3000 board). The leftmost switch, labelled “1,” is the Execute command which is activated by toggling the switch twice. The next three switches (labelled 2-4) select the opcode. Opcode encoding is shown in the table below. Use the right-most four switches (labelled 5-8) to input data. When the extended instruction set is selected with opcode 111, these switches provide additional bits of opcode.

Note: The rocker switches on the XC3000 demonstration board are On when down, Off when up. Use the “0” and “1” labels on the board as your guide.

Table 11-9 Processor Operations

2	3	4	5	6	7	8	Operation
0	0	0	*DATA*				ADD between switches and register
0	0	1	*DATA*				AND between switches and register
0	1	0	*DATA*				OR between switches and register
0	1	1	*DATA*				XOR between switches and register
1	0	0	*DATA*				SUB switch value from register
1	0	1	X	X	X	X	CLEAR register
1	1	0	*DATA*				LOAD register
1	1	1	0	0	0	X	ADD between stack and register
1	1	1	0	0	1	X	AND between stack and register
1	1	1	0	1	0	X	OR between stack and register
1	1	1	0	1	1	X	XOR between stack and register
1	1	1	1	0	0	X	SUB stack value from register
1	1	1	1	0	1	X	PUSH register value to stack
1	1	1	1	1	0	X	POP stack value to register
1	1	1	1	1	1	X	NOP

To perform an operation, set the data on the right-most “nibble.” “On” is a one; “Off” is a zero. Look up the correct opcode for the operation you want to perform and set the three opcode switches to the correct value. Then toggle the leftmost Execute switch twice. If the switch is already On, switch it Off, wait a moment, and then return it to the On position.

The contents of the ALU register are displayed in hexadecimal on the seven-segment display. The top value in the stack is displayed in binary on the bank of LEDs.

1. Verify that the initial contents of both ALU and stack are all zeros. The decimal display says “0,” and the LED bar is all Off. Now put a 1 on the data switches and load the switch value to the ALU register. The op code is 110.
2. Set the seven right-most switches to 110-0001.

3. Toggle the leftmost switch to execute the command. The decimal display shows the contents of the ALU register, which is now “1.” The stack is still empty. Add 13 to the ALU register. The opcode is 000.
4. Set the seven right-most switches to 000-1101.
5. Toggle the leftmost switch twice to execute the command. The decimal display shows the contents of the ALU register, which is now “E.” The stack is still empty. Push the register value onto the stack. The opcode is 111, which is the extended opcode. The data must be set to 101*x*, where the *x* is a don’t-care.
6. Set the seven right-most switches to 111-1011.
7. Toggle the leftmost switch twice to execute the command. The decimal display still shows “E.” The stack value is also “E,” so the LED bar shows 1110 in the right-hand nibble. Perform an XOR operation between the switch value and the register. The opcode is 011. Set the data to all ones.
8. Set the seven right-most switches to 011-1111.
9. Toggle the leftmost switch twice to execute the command. The decimal display changes to “1.” The stack value on the LED display is still “E,” 1110. Pop the value from the stack. The opcode is 111, which is the extended opcode. The data must be set to 110*x*, where the *x* is a don’t-care.
10. Set the seven right-most switches to 111-1101.
11. Toggle the leftmost switch twice to execute the command. The decimal display changes to “E.” The stack value returns to “0.” Clear the ALU register. The opcode is 101. The data is ignored.
12. Set the seven right-most switches to 101-1101.
13. Toggle the leftmost switch twice to execute the command. The decimal display changes to “0.” The stack value remains at “0.”
14. Try any other commands that you wish.

Making Incremental Design Changes

After initially placing and routing a design, it is often necessary to go back to the schematic and make slight modifications to the original design. When this situation occurs, much of the place and route information from the previous design iteration can be “recycled”, as much of it is unchanged. This process is known as incremental design, and the LCA file (containing partition, placement and routing information) from the prior place and route run is the guide file.

Since much of the place and route information is extracted from the guide file, the place and route time is greatly reduced. The reuse of place and route information also results in more stable timing over a number of guided place and route iterations. Once a section of your design passes your timing requirements, guided design ensures that it will pass in the future, even if other parts of the design are modified.

In this section of the tutorial, you make a small change to the schematic and reprocess the design using the guide option.

Note: A small design change is the addition, removal, or replacement of only a small amount of logic in the design; the exact amount is dependent on the size of the design. If radical changes are made to a design, it can be disadvantageous to guide the design.

Creating the Guide LCA File

First you must save the routed LCA file to another name. This file is used as the guide file. Open a shell in UNIX, make sure you are in the $xilinx_tutorial/calc_da directory, and type **`cp calc.lca gcalc.lca`** ↵.

The LCA file is not the only file that is used as input with the guide option. Incremental design with any member of the XC2000 or XC3000 families also requires a Partitioning Guide File (PGF). This file is described below. If you complete an XC2000 or XC3000 family design and want to use it as a guide file at a later time, save the PGF file as well as the LCA file. If necessary, the PGF file can be recreated using the LCA2XNF program. There is no PGF file for XC4000 family designs.

Making an Incremental Schematic Change

Make a simple change to the Calc schematic that will be visible immediately on the demonstration board. For example, assume that the reset opcode is no longer needed and needs to be removed from the design. This can be done by deleting the RST net on the top-level schematic. The logic that generated the RST signal, and the logic it drove, is automatically optimized out of the netlist by the XNFPrep program.

Open PLD_DA and load the Calc schematic.

1. Since you will return to XDM, do not close it, but iconize it on the desktop.
2. From PLD_DMGR, select the $xilinx_tutorial / calc_da / calc design object and choose **Right Mouse Button → Open → PLD_DA.** Design Architect appears with the Calc design loaded.
3. Use the F8 key to zoom in on the upper left quadrant of the schematic.
4. Press the F2 key to unselect everything.
5. Select RST net, connected between the CONTROL and ALU components, as shown in the figure below.
6. Press the Delete key to delete the net.
7. Check and save the schematic.
8. Exit PLD_DA and return to PLD_DMGR.

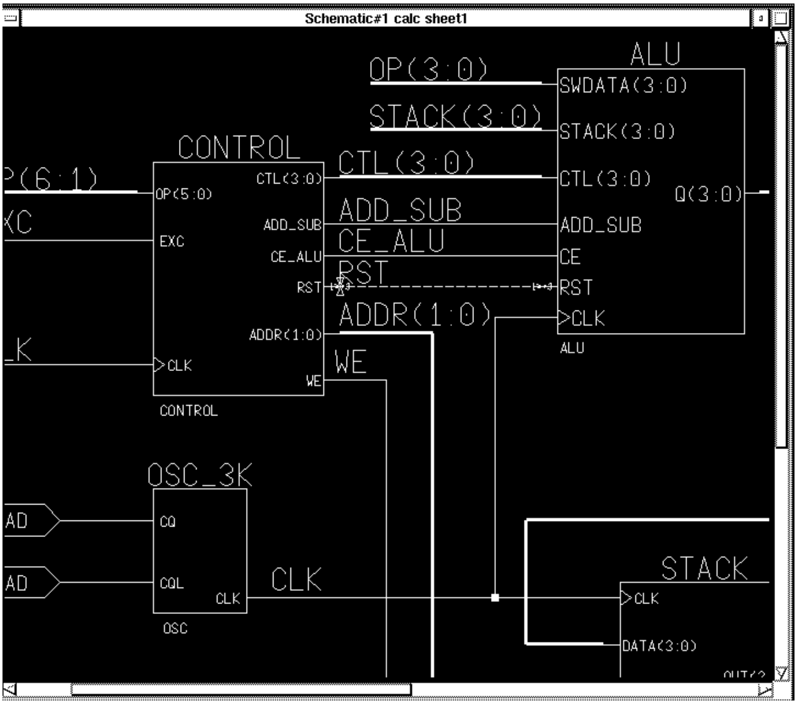


Figure 11-54 Disconnecting RST in Calc Schematic

Translating the Incremental Design

Translate the guided Calc design in exactly the same way as you did the first time.

1. Select the Calc design in the navigator window.
2. Since the design has been modified, it is necessary to generate a new netlist before running PLD_XMake. Choose **`Right Mouse Button`** → **`Open`** → **`PLD_Men2XNF8`**. Use the default options and select **`OK.`**
3. Now that the design changes are reflected in the netlist, select the calc.xnf file from the navigator and choose **`Right Mouse Button`** → **`Open`** → **`PLD_XMake`**. The PLD_XMake dialog box is displayed.

4. Enter the appropriate part type, choose the verbose and output to screen options as before.
5. Select **Yes** for Use Guide File? option. A File Name field appears.
6. Type **gcalc.lca** for the name of the guide file.
7. Press return or select **OK**.

The XMake program processes all of the necessary design files and displays its progress on the screen. It also automatically performs the steps necessary to guide the design.

If the design is targeted for an XC3000 family part, the partitioning of the design is done by XNFMap. To mimic the partitioning of the previous run, XNFMap reads the PGF file created by the previous run of XNFMap. Then, APR (or PPR for 3000A designs) reads the .LCA file so that it can follow the CLB placement and routing from the previous iteration. In 4000 family designs, PPR does all of the partitioning, placement, and routing, and the PGF file is not necessary. In either case, XMake automatically reads the appropriate files and performs the correct steps. If the translation is successful, XMake issues the following message:

```
’calc.bit’ has been made, check output in
’calc.out’
```

Checking for Errors in the calc.out File

Next, look at the OUT file created by XMake and check for errors. There should not be any errors or warnings that were not present in the initial OUT file. It is important to verify that the placement and routing completed successfully and that there are no unrouted nets.

For a description of the command flow followed by XMake for original and incremental translation, see the “Command Summaries” section at the end of this chapter. The command summaries are simplified, but valid, versions of the program flow used by XMake.

Verifying the Change in the Demonstration Board

Verify that the change was performed by downloading the new bitstream to the demonstration board, as you did previously.

1. Set all of the input switches High. This setting (SW5 switches High on the XC3000 / XC4000 and XC4000 boards; SW1 switches to “1” on the XC3000 board) selects the No-Op command.
2. Double-click on the XDM icon to open it.
3. In XDM, select the Verify menu.
4. Select **XCHECKER**. The XChecker software is used for either hardware cable. The port name is already set in the xchecker.pro file saved during the first download.
5. Select **Done** and the input file name: **CALC.BIT**.
6. If you are using the Download cable to program an XC4000 family part, press the PROG button. This step is not necessary if you are using the XChecker cable, or using the Download cable to program an XC3000 family part.
7. Press the ↵ key. If the LCA is successfully programmed, the following message appears:

   ```
   DONE signal went high.
   ```
8. Press any key to return to XDM.
9. If the Done signal does not go HIGH, check the connections between the cable and the demonstration board, power the board off and on, and try downloading again.
10. Verify that the change has been made by loading a value into the register, then attempting to execute the RESET (101XXXX) command.

Leaving XDM

1. To leave XDM, select **quit** or type **quit** ↵. The following message appears:

   ```
   Current profile option changes will be lost.
   Do you really want to quit (Y/N)?
   ```
2. You do not want to save the profile changes that set up XMake for incremental design. Click on **Yes** in the menu or type **y** ↵ to quit without saving your profile changes.

You have completed the Xilinx Design Architect tutorial. At this point, you can continue with the QuickSim II tutorial or skip to the X-

BLOX Tutorial, the Xilinx ABEL Tutorial, or the XACT-Performance and XDelay Tutorial.

Command Summaries

Although this tutorial uses XMake to process the Calc design, you can also manually run the individual programs that XMake runs. You can also bypass XDM and run either XMake or each individual program from the system prompt.

This section details command sequences that you can use to perform the translations XMake performs in this tutorial. The commands are written as you would type them at the system prompt or in a batch file. XMake executes some of these commands individually on each file in the design hierarchy, so that it does not rerun any program unnecessarily, but it is not necessary for you to do the same.

The cstfile options are necessary for specifying pad location constraints when compiling the Calc design. You may need to use them for your own designs.

Basic Translation for XC3000A and XC3000L Designs

men2xnf8 calc -p <part>	Translate schematics to XNF netlist
xnfmerge calc	Combine into one XNF file
xnfprep calc	Trim logic and check for errors
xnfmap calc	Map to CLBs and IOBs
ppr calc	Place and route
xdelay -d -w calc	Add delays to LCA file
makebits calc	Create bitstream

Basic Translation for XC4000 Family Designs

men2xnf8 calc -p <part>	Translate schematics to XNF netlist
xnfmerge calc	Combine into one XNF file
xnfprep calc	Trim logic and check for errors
ppr calc cstfile=calc_4k	Partition, place and route
xdelay -d -w calc	Add delays to LCA file
makebits calc	Create bitstream

Basic Translation for XC3000, XC3100, and XC2000 Family Designs

```
men2xnf8 calc -p <part> Translate schematics to XNF netlist
xnfmerge calc           Combine into one XNF file
xnfprep calc            Check for errors
xnfmap calc             Map to CLBs and IOBs
map2lca calc            Convert to LCA file
apr -w calc calc -c calc_3k.cst
                        Place and route, add delays
makebits calc           Create bitstream
```

Incremental Translation for XC3000A and XC3000L Designs

Make schematic changes.

```
copy calc.lca gcalc.lca  (Not run by XMake)
```

```
men2xnf8 calc -p <part> Translate schematics to XNF netlist
xnfmerge calc           Combine into one XNF file
xnfprep calc            Trim logic and check for errors
xnfmap -k calc          Map to CLBs and IOBs
ppr calc guide=gcalc    Place and route
xdelay -d -w calc       Add delays to LCA file
makebits calc           Create bitstream
```

Incremental Translation for XC4000 Family Designs

Make schematic changes.

```
copy calc.lca gcalc.lca (Not run by XMake)
```

```
men2xnf8 calc -p <part> Translate schematics to XNF netlist
xnfmerge calc           Combine into one XNF file
xnfprep calc ignore_locs=IO
                        Trim logic and check for errors
ppr calc guide=gcalc cstfile=calc_4k ignore_locs=IO
                        Partition, place and route
xdelay -d -w calc       Add delays to LCA file
makebits calc           Create bitstream
```

Incremental Translation for XC3000, XC3100, and XC2000 Family Designs

Make schematic changes.

```
copy calc.lca gcalc.lca (Not run by XMake)

men2xnf8 calc -p <part> Translate schematics to XNF netlist
xnfmerge calc           Combine into one XNF file
xnfprep calc            Check for errors
xnfmap -k calc          Map to CLBs and IOBs
map2lca calc            Convert to LCA file
apr -w -g gcalc calc calc
                        Place and route, add delays
makebits calc           Create bitstream
```

Further Reading

The Design Architect tutorial is provided to give you the information necessary to begin a Xilinx design using Mentor Graphics software. It is important to note that a tool as broad and complex as Design Architect cannot be fully explained in a single tutorial. There are many different ways to use the commands in Design Architect, and there are also many ways to customize the application. It is strongly recommended that you read the Mentor Graphics Design Architect documentation as well as the Xilinx Mentor Graphics Interface User Guide.

Mentor Graphics Interface/ Tutorial Guide

QuickSim Tutorial

Chapter 12

QuickSim Tutorial

This tutorial steps you through both a functional simulation and a timing simulation using the Mentor Graphics QuickSim II program. The Calc design created in the “Design Architect Tutorial” chapter or one of the completed designs in the solutions directories can be used for this tutorial. You can choose to target the design to any of the following parts: XC3020APC68, XC3020PC68, XC4003APC84, or XC4003PC84.

Note: Although this tutorial describes simulating FPGA designs, you can apply most of the steps to EPLD designs. See the “XEPLD Tutorial” chapter for EPLD-specific information.

Required Background Knowledge

This tutorial assumes that you have a basic understanding of the following:

- UNIX Operating System. None of the command sequences are given in AEGIS.
- Motif Windows. Mentor Graphics applications conform to the Motif window style.

Note: When you are instructed to close a window, it is important that you exit from the window rather than iconize it.

- Some knowledge of Design Manager, QuickSim II, and Xilinx core software. For more information on these applications, refer to the list of related publications at the beginning of this user guide.

Software Installation

Required Software

The following versions of software are required to perform this tutorial:

- Mentor Graphics Version 8.2_5 or later
- Xilinx / Mentor Graphics Interface DS344 Version 5 or later
- XACT Design Manager (XDM) Version 5 or later

Before Beginning the Tutorial

Before beginning the tutorial, set up your workstation to use Mentor Graphics and XACT Development System software as follows:

1. Verify that your system is properly configured. Consult the release notes that came with your software package for more information.
2. Install the following sets of software:
   - XACT Development System (DS501 or DS502) Version 5.00
   - Xilinx DS344 Mentor Graphics Version 5 interface
   - Mentor Graphics software Version 8.2_5 or later, including Design Manager, Design Architect, QuickSim II, QuickPath, as well as the software needed to produce EDIF netlists from ENWRITE, which requires special licensing
3. Verify the installation. When you finish the installation, verify that your .cshrc or setup file contains lines similar to the following:

Note: Path names of directories will vary. For more information on paths and environment variables, refer to your Release Notes.

```
setenv LCA /location_of_ds_344:
setenv XACT /location_of_ds_344:
/location_of_ds502
set PATH=($PATH                  \
   $LCA/com/sparc                \
   $LCA/bin/sparc                \
  /location_of_ds502 /bin/sparc  \
   )
```

Modifying Mentor Graphics Variables

Make sure that the following Mentor Graphics specific variables are set correctly:

- MGC_HOME

  This should point to the Mentor Graphics software tree.

- MGC_GENLIB

  This should point to the Mentor Graphics gen_lib library, normally $MGC_HOME/gen_lib.

- LD_LIBRARY_PATH

  This variable is used by the Mentor Graphics Design DataPort (DDP) routines that are accessed by some Xilinx programs. On a SPARC station with OpenWindows installed in /usr, this variable is set as follows:

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

- MGC_LOCATION_MAP

  This variable should point to a valid location map file.

  Every symbol and schematic in your design contains a reference. A reference indicates where the design object resides on your disk. The tutorial designs use variables in their reference definitions so they can be easily relocated. All of the tutorial designs use the variable, $xilinx_tutorial, to define the path reference. $xilinx_tutorial must be defined in the file pointed to by $MGC_LOCATION_MAP. For example, the design object, led_inv in the *install_path*/tutorial/mentor/calc_da directory, uses the path reference $xilinx_tutorial/calc_da/led_inv to define where it is located in the directory structure. If the tutorial directories were copied to the path, /home/bclinton/mentor/xtutorial, the following two lines must be added to the file pointed to by $MGC_LOCATION_MAP:

  ```
  $xilinx_tutorial
  /home/bclinton/mentor/xtutorial
  ```

  If a query was made to determine where the design object '$xilinx_tutorial/stack' is located, the Mentor Graphics tools would use this definition to determine that stack is at /home/bclinton/mentor/xtutorial/calc_da/stack.

It is also important that the $LCA variable be instantiated, but not defined, in the file pointed to by $MGC_LOCATION_MAP. To do this, add the following line to MGC_LOCATION_MAP, followed by an empty line:

LCA
(empty line)

Refer to the Mentor Graphics documentation for more information on location maps.

- MGC_WD

  This variable should point to the working directory. For the tutorial, it should point to the directory where the tutorial is worked on.

- LCA

  In addition to instantiating it in the file pointed to by MGC_LOCATION_MAP, the LCA environment variable should point to the directory where the DS344 software is installed.

Installing the Tutorial

If you have performed the Design Architect tutorial, stay in the same project directory and continue with the next section, “Basic Functional Simulation”.

If you choose not to perform the Design Architect tutorial, you must perform the following steps before you can begin the QuickSim tutorial.

1. The tutorial files are optionally installed when you install the DS344 interface software. If you have already installed the software, but are not sure whether you specified tutorial installation, check for check for a tutorial directory under your DS344 directory. The tutorial directory contains the tutorial files.
2. Since the schematics used in the Design Architect tutorial are incomplete when installed, you must copy a completed set of schematics and symbols from one of the solutions directories.You must also copy the routed calc.lca file so that you can do timing

simulation. Solutions for the QuickSim tutorial are supplied in the following directories:

calc_3k — solution files for XC3020PC68
calc_3ka — solution files for XC3020APC68
calc_4k — solution files for XC4003PC84 and XC4003APC84

It is recommended that you copy the appropriate tutorial directory in its entirety from the original install area to another area as described in a later step. This is done using the copy command in PLD_DMGR. Never use the UNIX cp command to copy a Mentor Graphics design.

Note: In this tutorial, file names and directory names are in lower case and the design example is referred to as Calc.

Starting PLD_DMGR

PLD_DMGR is the Mentor Graphics Design Manager run with a special start-up script that adds Xilinx icons to the tools window. To invoke PLD_DMGR, perform the following steps:

1. In UNIX, go to the project directory.
2. Use the UNIX command, setenv, to set the $MGC_WD to the directory where you will be working on the tutorial. For example, if you plan on copying the calc_4k solutions directory from the DS344 install area to the directory /home/dum/mentor/tutor, type the following:

   ```
   setenv MGC_WD /home/dum/mentor/tutor/calc_4k
   ```

3. Invoke PLD_DMGR by typing **`pld_dmgr`**.

Note: It is important that $MGC_WD be set to the working directory. Problems may occur in the Xilinx scripts if this variable is not set correctly.

Making a Working Copy of the CALC design

Note: If you already have a Calc design from completing the Design Architect tutorial, you can use it for this tutorial if desired. Skip the following steps and go to the next section.

Perform the following steps to make a working copy of the Calc design:

1. In the navigator, select from the tutorial/mentor directory in the DS344 installation area the design directory you want to use (either calc_3k, calc_3ka, or calc_4k). Pick calc_3k for an XC3000 design, calc_3ka for an XC3000A design, or calc_4k for an XC4000 family design.
2. Select **RIGHT MOUSE BUTTON → EDIT→ COPY.** A dialog box appears.
3. In the dialog box, type the path to the directory where you want the working copy of the tutorial files to reside. For example, if you want to copy the calc_4k design directory to /home/dum/tutor, type "/home/dum/tutor/calc_4k" in the dialog box.
4. Select **OK** to execute the Copy command.
5. Use the navigator to change directories until the newly-created working copy of the tutorial directory appears in the navigator.

Basic Functional Simulation

The Calc design, at this point, does not contain any X-BLOX or Xilinx ABEL elements, so very little pre-processing is necessary for simulation. Refer to the appropriate tutorial chapter for information on simulating designs with Xilinx ABEL or X-BLOX elements.

Preparing the Calc Schematic for Simulation

Components in the Calc design at this point have built-in simulation models so little pre-processing is necessary. However, every design must have a simulation viewpoint before it can be used in QuickSim. The viewpoint defines the simulation model that should be used for a primitive. The scripts, Men2XNF8 and FNCSIM8, are represented as the PLD_Men2XNF8 and PLD_FNCSIM8 icons in PLD_DMGR.These icons are used to prepare a design for functional simulation.

You must run Men2XNF8 on a design before functional simulation because FNCSIM8 reads the netlist produced by Men2XNF8 to determine how much pre-processing is required. An X-BLOX design, for example, requires more pre-processing than a design without X-BLOX symbols.

PLD_Men2XNF8

The design is translated by the Men2XNF8 program to an XNF (Xilinx Netlist Format) file. Invoke Men2XNF8 on the Calc design using the following method:

1. Select the Calc design object from the appropriate directory in the navigator window.
2. Invoke Men2XNF8 on the design by selecting **RIGHT MOUSE BUTTON → OPEN → PLD_MEN2XNF8**. A dialog box appears.
3. Type the appropriate part type in the Part Type field, then select **OK.** The Men2XNF8 script executes. If errors occur, check the men2xnf8.log file for more information.

PLD_FNCSIM8

Invoke FNCSIM8 on the Calc design using the following method:

1. Select the Calc design object in the navigator window.
2. Invoke FNCSIM8 on the design by selecting **RIGHT MOUSE BUTTON → OPEN → PLD_FNCSIM8**. A dialog box appears.
3. Select **Use Original**. This specifies that the original schematic is used in simulation.
4. Select the **Yes** for Run QuickSim, then press return or select **OK**. The FNCSIM8 script executes and a simulation viewpoint is generated. The design is then automatically loaded into QuickSim II.

Viewing the Calc Schematic

When QuickSim starts, no windows are open. Open a window and view the top-level schematic for the Calc design. Displaying the schematic is convenient for viewing back-annotation during the simulation.

1. To open a window containing the Calc schematic, select **Open Sheet** from the palette. This automatically opens the top-level sheet for Calc.
2. Move the window to the upper left corner of the QuickSim window.

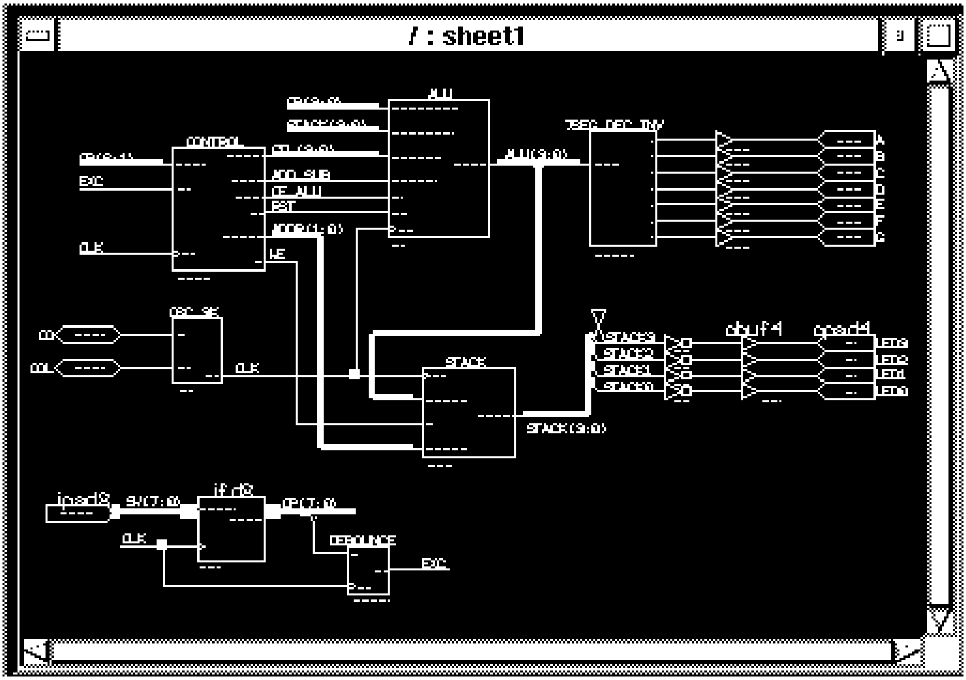


Figure 12-1 Top-level Calc schematic

Selecting Nets for Simulation

There are several ways to select the signals that you would like to monitor. One way is to select the **Right Mouse Button → Add → Traces → Specified** command, then type in the nets you want to view in simulation. Another way is to select the nets on the schematic. To select the signals, you may need to zoom and pan using the scroll bars or using strokes.

To select the nets on the schematic:

1. Using the F8 key, zoom in on the area pictured in the following figure.
2. Position the cursor on the net labeled CLK, and press the left mouse button. The net appears highlighted, as in the figure below. Whenever any portion of a net is selected in QuickSim, the entire

net appears highlighted. You can select additional nets using the same procedure. If you make a mistake, click the left mouse button a second time on the net or object to unselect it.

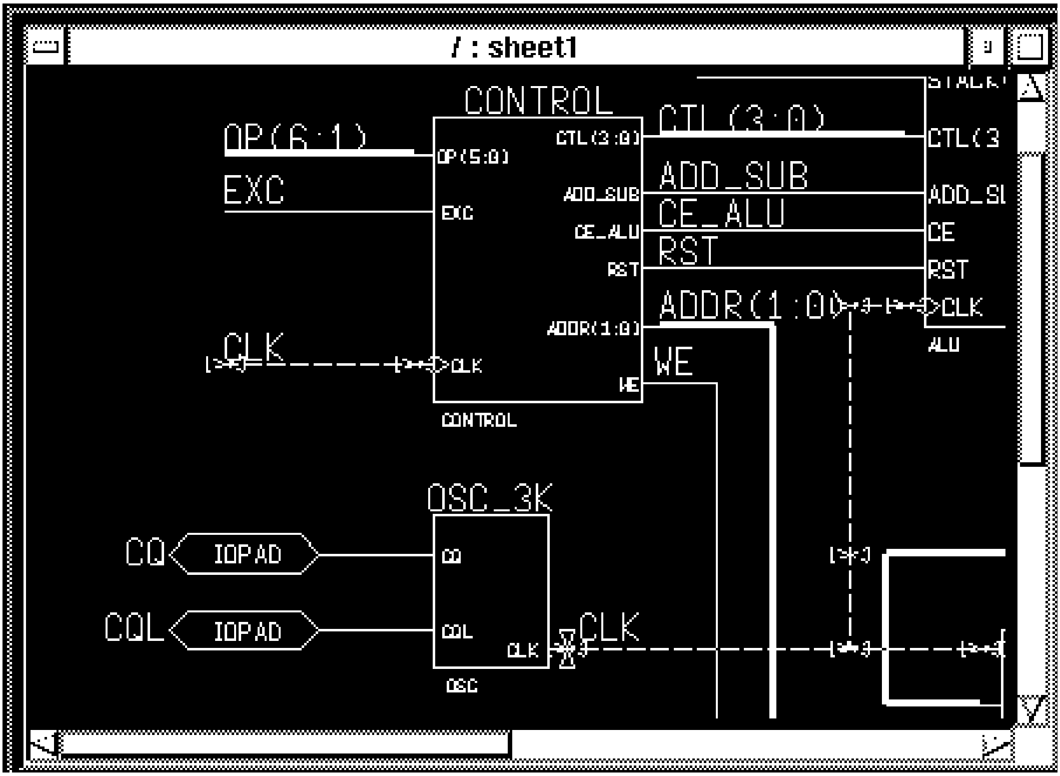


Figure 12-2 Selecting the CLK Net for Display in Trace Window

3. Use the left mouse button to select the following nets: WE, RST.
4. Using the Shift -F8 key, view the entire schematic.
5. Select the net labeled EXC (output of the DEBOUNCE component) with the left mouse button.

   One of the advantages of labeling all nets is now clear. When you select an unlabeled net for simulation display, note that a default name is used for the net, such as N$14. This name is not very useful for debugging, especially since making changes to the schematic may cause renumbering of net names.

6. You can also add buses to your list of signals to be monitored. Use the left mouse button to select the buses labeled ALU(3:0) and STACK(3:0).
7. Press the blue TRACE button in the palette to add these two buses to the trace window.

A bus selected in this manner is displayed as a bus in the Trace window. Later in the tutorial you learn how to create buses out of discrete signals on the schematic, so that they may be viewed as a bus in the Trace window.

Opening Trace and List Windows

To view the waveforms of the selected signals, you must open a Trace window. To open a trace window, select the blue button labeled **`Trace`** in the palette with the left mouse button.

A Trace window appears. The waveforms selected on the schematic appear in the Trace window. It may be necessary to resize the Trace window to see all the signals at once. Otherwise, move the cursor into the Trace window and use the PageUp and PageDown keys to scroll the signals in the window.

Note that all the signals in the Trace window are highlighted. Every window opened in QuickSim is dynamically linked to the others. The selection of a net on the schematic sheet, for example, is also reflected in the Trace window, and in any other window that is open. This is useful, for example, if a setup violation occurs. The instance name in the error message text will be highlighted, and the related component on the schematic page will also appear highlighted.

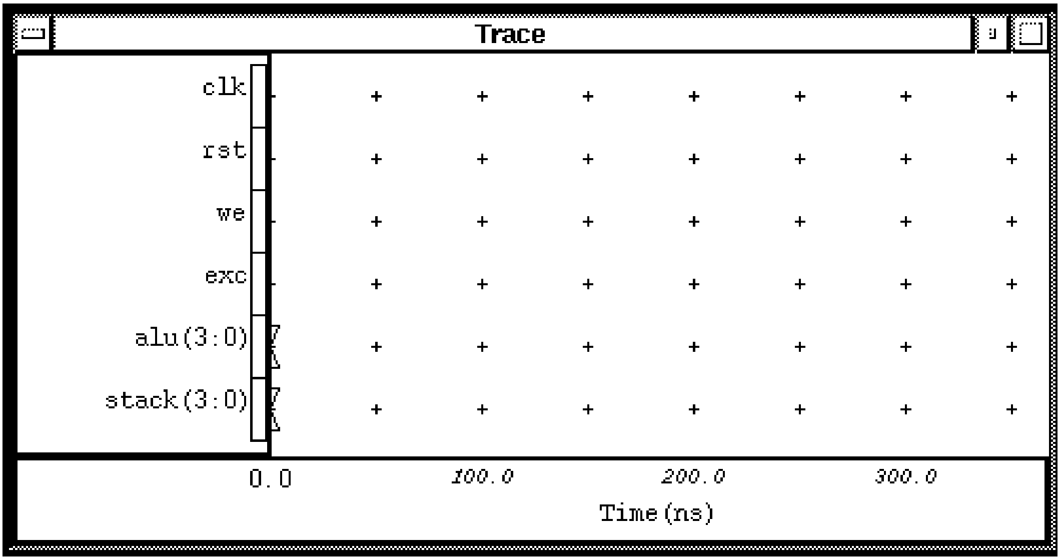
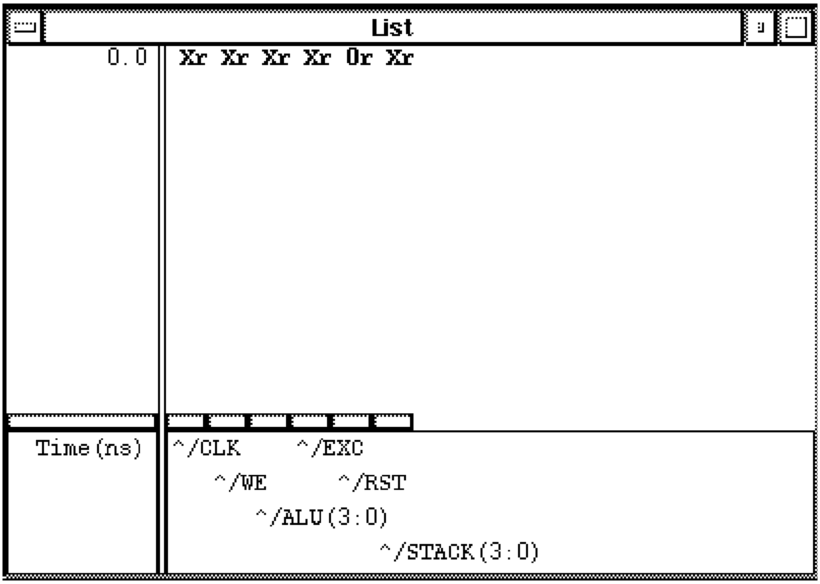


Figure 12-3 Trace Window

It is sometimes useful to obtain tabular output using a List window. A List window displays signal values and highlights the points at which a given signal value changes. To open a List Window, perform the following:

1. Since the desired signals are already selected, select the blue **LIST** button in the palette with the left mouse button. The list window appears, with the signal names at the bottom. The caret (‘^’) is an arrow pointing up to indicate the correct column for each signal.
2. Move the List window to the upper right-hand corner of the screen next to the schematic window.



```
        0.0  Xr Xr Xr Xr 0r Xr

   Time(ns)  ^/CLK    ^/EXC
                ^/WE     ^/RST
                   ^/ALU(3:0)
                          ^/STACK(3:0)
```

Figure 12-4 List Window

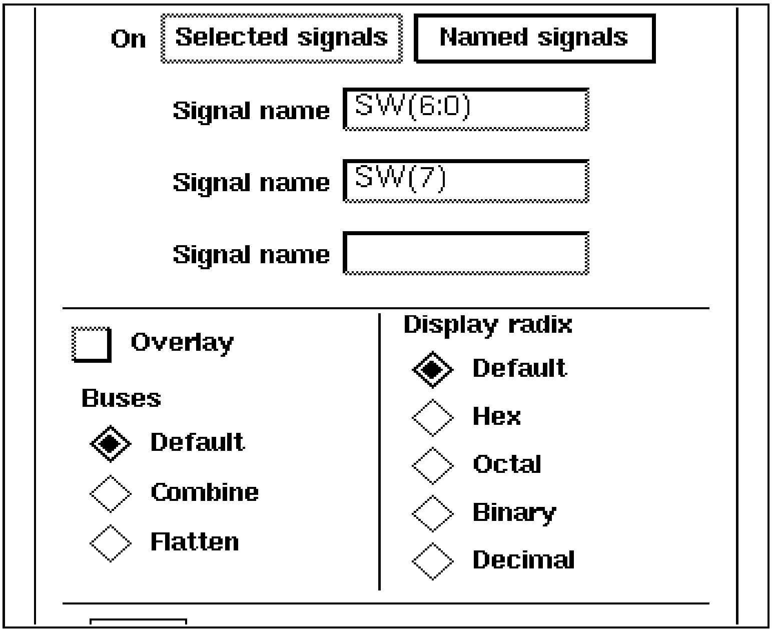
Adding Traces Manually

In the Calc design, inputs are entered via a set of eight switches, SW(7:0). The lower seven switches (SW(6:0)) define the opcode. The left-most switch (SW(7)) is the execute switch. When SW(7) is toggled, the selected opcode on SW(6:0) is executed. It is useful to view SW(6:0) and SW(7) separately in the Trace window.

Add these two traces to the Trace window as follows:

1. Press the F2 key to unselect everything.
2. Select the Trace window with the left mouse button.
3. Choose **Right Mouse Button → Add → Traces → Specified...** A dialog box appears.
4. Select the Named Signals button in the dialog box with the left mouse button. A field for entering text appears.
5. Fill in the dialog box as shown in the figure below.
6. Select **OK** or press return. The bus SW(6:0) and the signal SW(7) are

added to the trace window.



Signal name SW(6:0)

Signal name SW(7)

Signal name

□ Overlay

Buses

◆ Default

◇ Combine

◇ Flatten

Display radix

◆ Default

◇ Hex

◇ Octal

◇ Binary

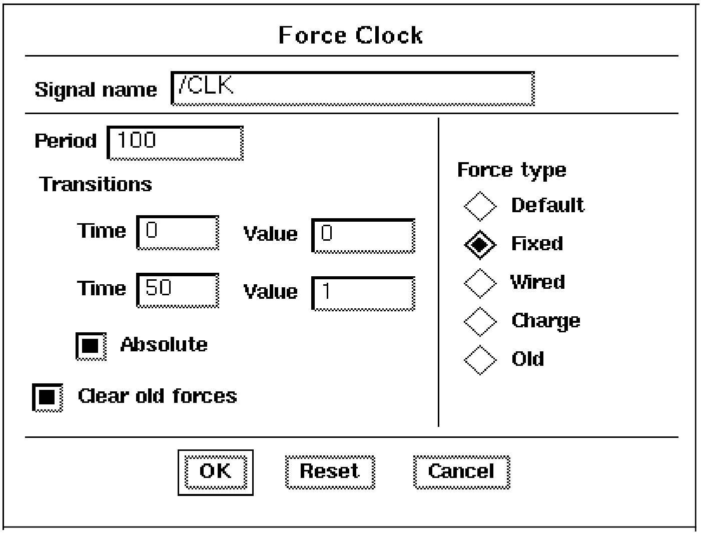
◇ Decimal

Figure 12-5 Adding Manual Traces

Assigning Values to the Clock

The first step is to define a clock for the circuit as follows:

1. Make sure that the trace window is active (border appears blue). If not, select the window using the left mouse button.
2. Press the F2 key to unselect everything, then select the CLK net in the Trace window using the left mouse button.
3. Select the red button labeled **STIMULUS** in the Palette. The icons in the Palette change.
4. Select **ADD CLOCK** in the Palette. A dialog box appears.
5. Fill in the dialog box as shown in the figure below.



Force Clock

Signal name /CLK

Period 100

Transitions

Time 0 Value 0

Time 50 Value 1

☒ Absolute

☒ Clear old forces

Force type

◇ Default

◆ Fixed

◇ Wired

◇ Charge

◇ Old

Figure 12-6 Adding Clock Waveform

The dialog box selections give the clock a 100 ns period and a 50 percent duty cycle. At zero ns, the clock begins with a value of zero. The Absolute option indicates that the times are absolute, and not relative to the state of the simulator. For example, if you had already been simulating, the state of the simulator may not be at zero ns. If Absolute is selected, the times entered in the dialog box are referenced from time zero. If Absolute is not selected, the times entered in the dialog box are added to the present time in the simulation.

Selecting a Fixed Force type indicates that the signal is driven as if it were connected directly to VCC or GND. If Wired were selected, the signal would be driven as if it were connected to a pull-up or pull-down resistor. A Charge Force type represents a default charge on a floating signal. Wired values are overridden by Fixed values, and Charge values are overridden by both. In general, for Xilinx designs always use a force type of Fixed unless it is a bidirectional input, in which case a Wired force type should be used.

6. Press return or select **OK** to add the force to CLK.

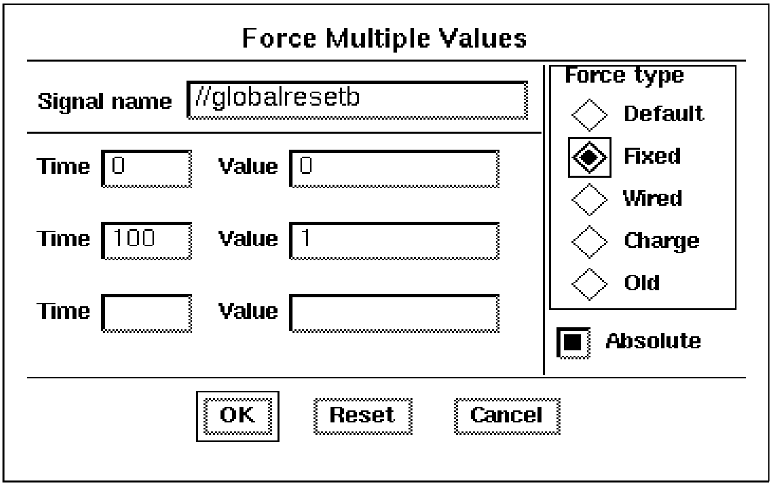
Asserting Global Reset (XC2000 & XC3000 Families Only)

This section applies only to XC3000 and XC2000 family designs. If your design is targeted toward an XC4000 family device, go to the next section, “Asserting Global Set Reset”.

The first node you must assert is the globalresetb signal. This signal does not exist on your schematic, but it is does exist in the device. This dedicated net is connected to the Reset pin on every XC2000 family or XC3000 family FPGA. It is an active-low signal that resets all of the flip-flops and IOB latches in the chip.

Globalresetb is part of the simulation models; you must toggle it at the beginning of every simulation. If you do not pulse globalresetb low, all flip-flop outputs are unknown at all times during your simulation. Set globalresetb low at time 0 and high at 100 ns as follows:

1. With the Trace window selected, press the blue **`Unselect All`** button in the Palette with the left mouse button.
2. Select the **`Add Force`** icon in the Palette with the left mouse button. A dialog box appears.
3. Since a signal is not selected, the Signal name field is empty. Fill in the dialog box as shown in the following figure. The globalresetb signal is entered with two leading forward slashes because it is a global signal. The signal is to be forced low (asserted) at time zero and high at time 100ns.



Force Multiple Values

Signal name: //globalresetb

Time	0	Value	0
Time	100	Value	1
Time		Value	

Force type:
- ◇ Default
- ◆ Fixed
- ◇ Wired
- ◇ Charge
- ◇ Old

■ Absolute

Figure 12-7 Forcing Globalresetb (XC3000 only)

Asserting Global Set Reset (XC4000 Family Only)

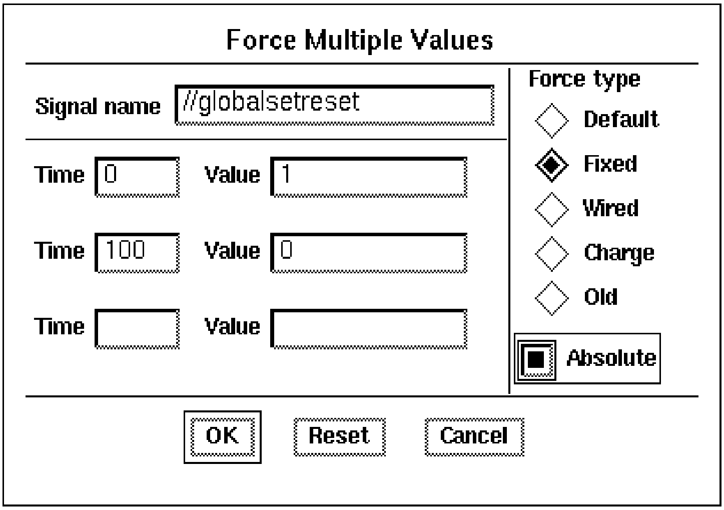
Note: This section applies only to XC4000 family designs. If your design is targeted toward a device from any other family, go to the “Design Description” section.

The Global Set Reset signal must be forced at the beginning of all XC4000 family simulations. It is an active-High signal that sets or resets all flip-flops in the chip. Whether a flip-flop is set or reset depends on whether it is an FDR or an FDS flip-flop, or on the value of the flip-flop’s INIT attribute. The default configuration for all flip-flops is to function as a reset flip-flop.

Unlike the XC3000 family devices, the globalsetreset signal is not hard-wired to a package pin, and need not appear on one at all. If you want access to the Global Set Reset net from an external pin, place the STARTUP component in your schematic and attach an IPAD and IBUF to the GSR pin. This pad becomes an active-High Global Set Reset signal. You can also use an internally generated signal to drive the GSR pin of the STARTUP component. There is also an active-High Global Three State signal (GTS) that you can access in the same way. See the *XACT Libraries Guide* for more information on the STARTUP symbol.

Globalsetreset is the name of the net in simulation. It is part of the simulation models, and must be toggled at the beginning of every simulation. If you do not pulse globalsetreset high, all flip-flop outputs are unknown at all times during your simulation. Set it high at time 0 and low at 100 ns as follows:

1. With the Trace window selected, press the blue **`Unselect All`** button in the Palette with the left mouse button.
2. Select the **`Add Force`** icon in the Palette with the left mouse button. A dialog box appears.
3. Since a signal is not selected, the Signal name field is empty. Fill in the dialog box as shown in the following figure. The globalsetreset signal is entered with two leading forward slashes because it is a global signal. The signal is to be forced high (asserted) at time zero and low at 100ns.



Force Multiple Values

Signal name: //globalsetreset

Time: 0 Value: 1

Time: 100 Value: 0

Time: Value:

Force type: ○ Default ● Fixed ○ Wired ○ Charge ○ Old

☒ Absolute

Figure 12-8 Forcing Globalsetreset (XC4000)

Design Description

The Calc design is a simple four-bit processor with a stack. The CONTROL module interprets the switch input and drives the control lines of the ALU and STACK components. The ALU performs functions between an internal register and either the top of the stack or data read in from the external switches. Outputs include ALU(3:0), the current contents of the internal register, and STACK(3:0), the top value in the stack.

For a more detailed description of the Calc design, see the “Design Architect Tutorial” chapter. Additional information, including a table of opcodes, appears in the “Testing the Design” section of the same chapter.

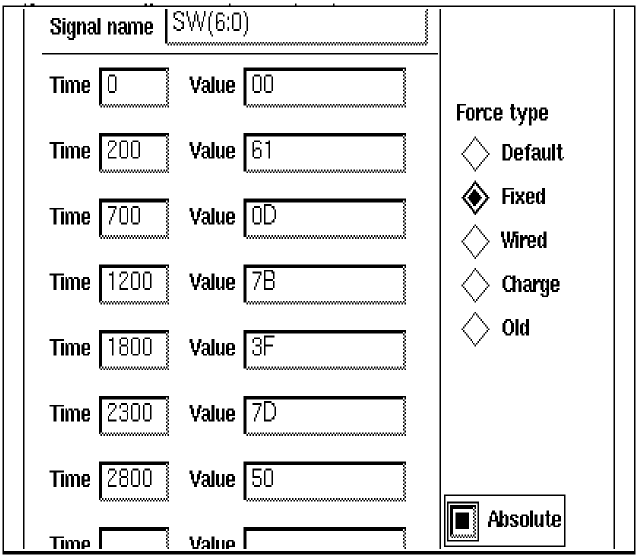
Simulating the Circuit

You are now ready to force the inputs to known values and simulate.

1. Press the blue **`Unselect All`** button in the Palette.
2. Select the **`Add Force`** button in the Palette with the left mouse button.
3. Fill in the dialog box as shown in the figure below.

   All numbers entered are interpreted as hex. This sets opcode (SW(6:0)) to perform the following actions:

   00: ADD 0h to register value (should produce a zero).
   61: LOAD register with 1h.
   0D: ADD Dh to register value (1 + D should produce F).
   7B: PUSH register value to stack (top of stack=F).
   50: CLEAR register value.



Signal name SW(6:0)

Time	Value
0	00
200	61
700	0D
1200	7B
1800	3F
2300	7D
2800	50

Force type

◇ Default
◆ Fixed
◇ Wired
◇ Charge
◇ Old

■ Absolute

Figure 12-9 Forcing Values to SW(6:0)

For these commands to be executed, you must provide stimulus to SW(7), the execute switch. Perform the following actions to force SW(7) correctly:

1. Press the blue **`Unselect All`** button in the Palette.
2. Select the SW(7) signal from the Trace window. It may be necessary to use the PageDown key to scroll through the list of signals in the Trace window.
3. Select the red **`WF EDITOR`** button from the top of the Palette. The icons in the Palette change.
4. Select the icon labeled **`EDIT WAVEFORM`**. A new trace appears labeled **`forces@@/SW(7)`**. While the **`SW7`** trace represents the value of SW7 up to the present time in simulation, the trace **`forces@@/SW(7)`** represents all values that will ever be forced on the signal. During simulation, this waveform can be edited to modify future values of SW(7).

A blue line appears extending from SW(7) to indicate that it has not been given a value. First, force SW(7) to a known value at time zero as follows:

1. Select the **CHANGE VALUE** icon in the Palette.
2. Move the cursor into the Trace window. A red vertical line appears under the cursor. The numbers in the grey box reflect the value and time that are pointed to as the cursor is moved.
3. Move the cursor close to the beginning of **forces@@/SW(7)**, as shown in the figure below, and then press the left mouse button. This indicates that you want to change the value from the nearest left edge (in this case, time zero is considered an edge) to the next right edge. Since the signal makes no transitions, you can assign the same value to the entire length of the signal.
4. Type a ‘1’ in the value field of the small dialog box and choose **OK.** This indicates that you want to change the signal value between the two nearest edges to a one. The entire length of the signal changes color from dark blue to light blue, and the line moves up, indicating it will be driven to a one.

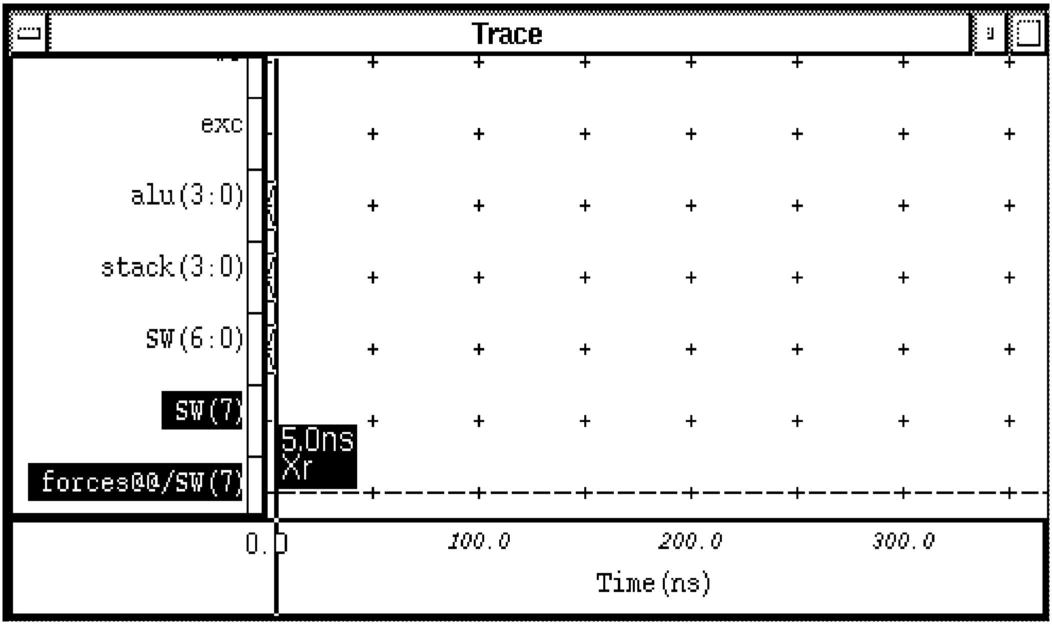


Figure 12-10 Forcing SW(7) to Initial Value

5. Press the Escape key to end the Change Value operation.

Now that SW(7) has been given an initial value, you must define when transitions occur on the signal as follows:

1. Select the **ADD TOGGLE** icon from the Palette.
2. Move the cursor to the trace window. A red vertical line appears with numbers indicating the value and time of the signal at the position beneath the cursor.
3. Move the cursor to the **forces@@/SW(7)** signal at time 700ns and press the mouse button, as shown in the figure below. A high to low transition is added to the force waveform at time 700ns.

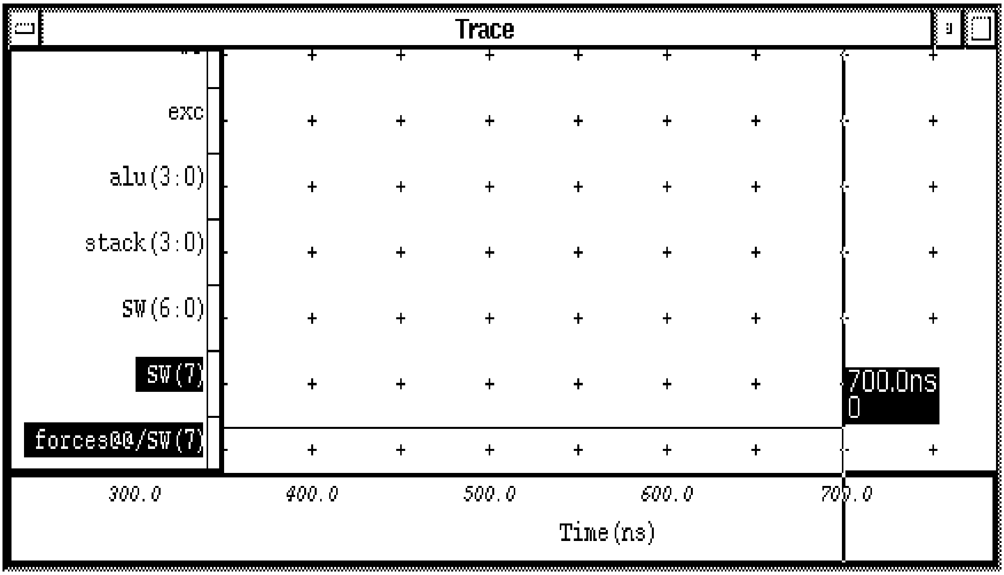


Figure 12-11 Adding the First Toggle to SW(7)

Note: It is sometimes difficult to position the cursor at exactly the right value if you are zoomed in too close. If you zoom out, the numbers get rounded to the nearest 1.0 ns, making it easy to place the edges correctly. Use the stroke 753 to zoom out. If you still cannot place the edges exactly, err to the left of the desired location. If you make a mistake, select the **CUT EDGE** icon in the Palette and click the left mouse button on the incorrectly placed edge. The edge disappears. Then, select **ADD TOGGLE** to continue adding edges.

4. Without moving the cursor, use the right arrow key to scroll the window forward in time. Each press of the right arrow key advances the window (and, consequently, the position under the cursor) by 50 ns. Add toggles at times 900, 1200, 1400, 1800, 2000, 2300, 2500, 2800, and 3000 ns. The waveform then appears as in the figure below. Press Shift-F8 to view the entire waveform.

5. Press Escape to end the **ADD TOGGLE** command

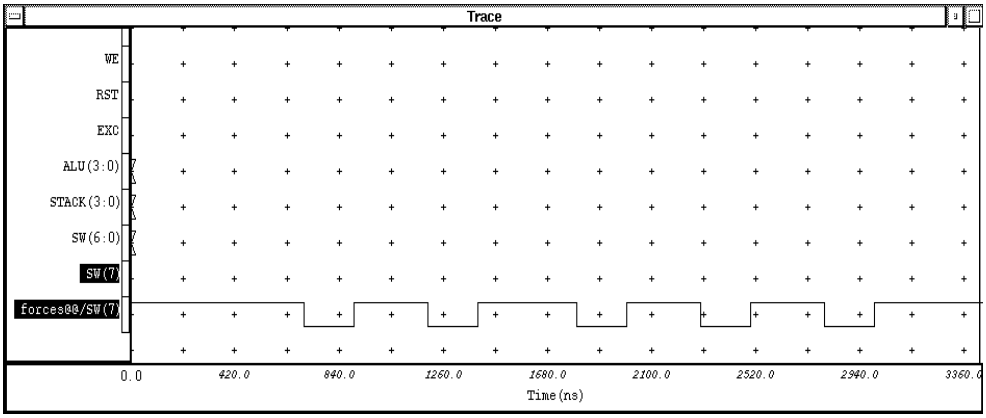


Figure 12-12 SW(7) Force Waveform

Now that your inputs and clock have been defined, you are ready to run the simulation.

Type **`run 3400`** at any location in the QuickSim window, then press return. A window automatically appears containing the text. The results should look similar to those in the following figure.

Note: XC4000 users will see unknown values on the STACK(3:0) output for parts of the simulation. Since the XC4000 stack is implemented using RAM, not flip-flops, it can only be initialized by writing to it. Globalsetreset does not initialize RAM.

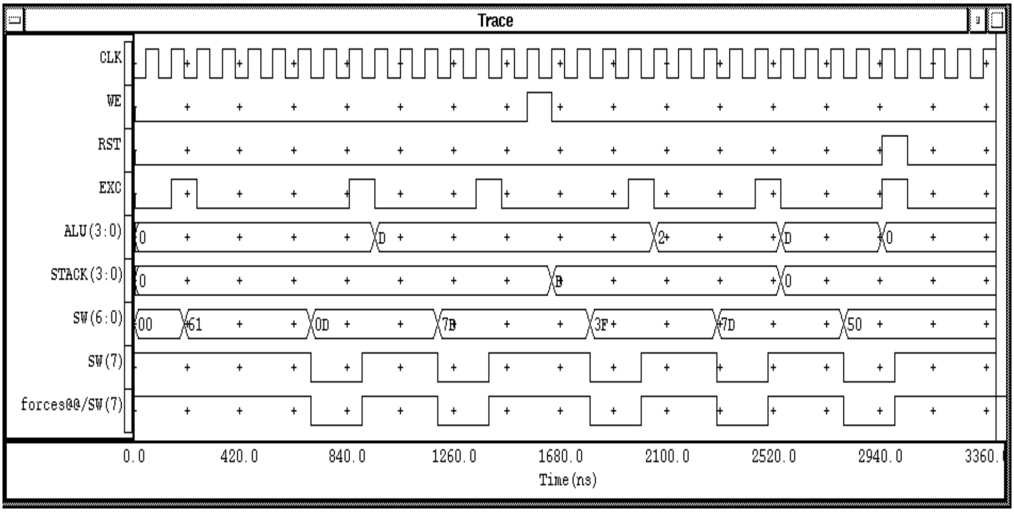
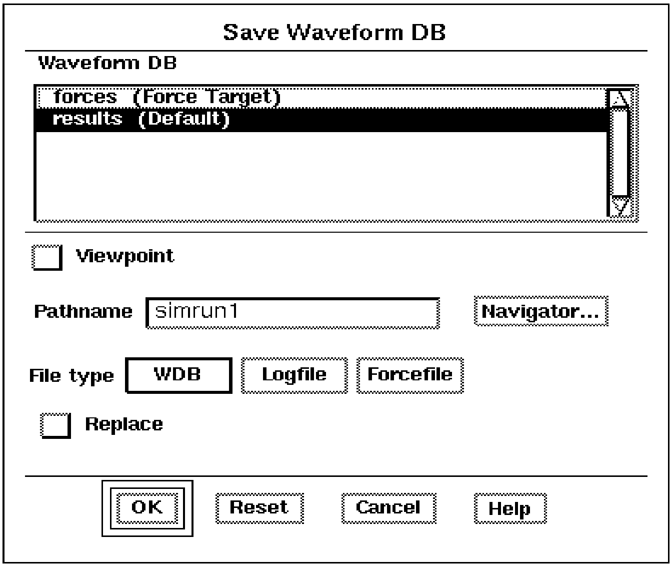


Figure 12-13 Output from Simulation (XC3000A design)

Saving the Results

If you were to exit QuickSim now, you would lose your waveform data. You can save the waveform information in a waveform database. To view the waveforms at a later time, you can use the **File → Load → Waveform DB** command found in the menu bar.

1. Select the red **STIMULUS** button from the palette.
2. Select the **SAVE WDB** icon from the Palette. A dialog box appears.
3. Fill in the dialog box as shown in the following figure. This saves your results to the WaveForm Database, simrun1. This database is created in the directory specified by the $MGC_WD environment variable.
4. Press return or select **OK**.



Save Waveform DB

Waveform DB

forces (Force Target)
results (Default)

□ Viewpoint

Pathname simrun1

File type: WDB

□ Replace

Figure 12-14 Saving Results

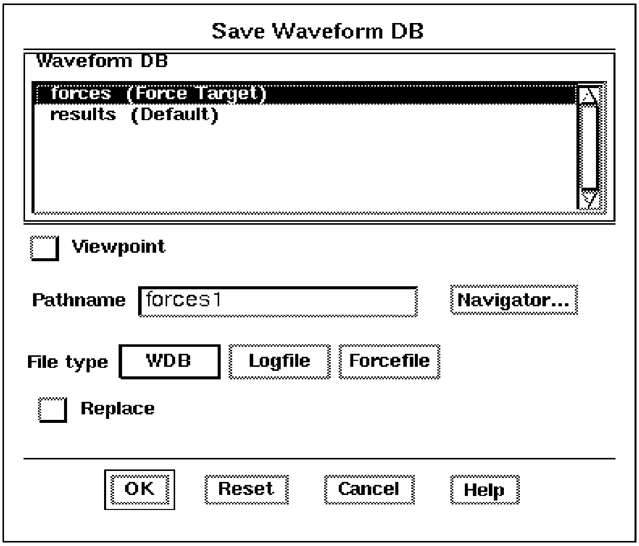
After saving the results, reset the simulator to time zero as follows:

1. Press the blue reset button in the palette. A dialog box appears.
2. Select the **`State`** button so that it highlights. Deselect any highlighted buttons. This forces the simulator to reset without saving.
3. Press return or choose **`OK`**. The trace window results disappear, while the forces waveform remains.

It may be useful to save the stimulus so that it can be run again. To do this perform the following steps:

1. Press the red **`STIMULUS`** button in the palette.
2. Select the **`SAVE WDB`** icon from the palette. A dialog box appears.
3. Fill in the dialog box as shown in the figure below. This saves the stimulus to the file, forces1. As with simrun1, this file is created in the directory specified by $MGC_WD.

4. Press return or choose **OK** to save the forces.



Save Waveform DB

Waveform DB

forces (Force Target)
results (Default)

□ Viewpoint

Pathname: forces1

File type: WDB

□ Replace

Figure 12-15 Saving Forces

Using the Transcript

In addition to saving the results and forces, you can also save the actual transcript for the QuickSim session. Every mouse click and key press is recorded. This is sometimes useful for making macros to perform complicated, repetitive tasks. The saved transcript can then be replayed using the **MGC → Transcript → Replay** command found in the menu bar. Save the transcript as follows:

1. Select the **MGC → Transcript → Show Transcript** command from the menu bar. A text window appears. In this text window are AMPLE commands. AMPLE is a C-like programming language used by all of the Mentor Graphics tools.
2. Select **Right Mouse Button → Export**. A dialog box appears.
3. Type the file name ‘transcript.out’ in the text field of the dialog box. This saves the transcript to that file.

   It is usually necessary to edit the transcript to make it useful. For example, if this transcript were re-run on the Calc design, it would setup the simulation, run, save the results, reset the simulator, and open a transcript window. Perhaps all you want it to do is setup and run the simulation. You would then have to delete the other commands from the transcript file before re-running it. For example, the $show_transcript(); command at the end of the transcript file could be deleted to keep the transcript window from appearing. You would probably also want to delete the $set_active_window("Transcript"); command as well if you did this. For more information on AMPLE, refer to Mentor Graphics documentation.

4. Select **File→Quit** to exit QuickSim.

Performing a Timing Simulation with PLD_TIMSIM8

The simulation flow previously described in this tutorial is for functional simulation. Timing simulation uses the block and routing delay information from the routed design to give a more accurate assessment of the behavior of the circuit under worst-case conditions. In this section, you perform a timing simulation of the Calc design by preparing the design using PLD_TIMSIM8.

The timing flow using PLD_TIMSIM8, as described in this section, works for simulating designs created with the Xilinx Unified Libraries or with the V1.10 libraries. The Calc design is composed of only standard library components, so the original schematic can be used for timing simulation. This timing flow is not applicable to designs with X-BLOX symbols, Xilinx ABEL symbols, MemGen symbols, or symbols that reference an XNF file. For more information on simulating these designs, see the applicable tutorial chapter.

Using PLD_TIMSIM8 to Prepare for Timing Simulation

PLD_TIMSIM8 reads a routed LCA file and back-annotates the delays to the schematic. This includes a number of steps, all of which are automatically run by the TIMSIM8 script. This script is represented by the PLD_TIMSIM8 icon in PLD_DMGR. The files necessary for back-annotation have either been created in the Design Architect tutorial or are included in the solutions directories.

Use PLD_TIMSIM8 to prepare the design for timing simulation as follows:

1. In PLD_DMGR, use the navigator to find and select the Calc design that you are using.
2. Select **Right Mouse Button → Open → PLD_TIMSIM8**. A dialog box appears.
3. In the dialog box, select **Use Original** radio button.
4. Select **Yes** for **Verbose Output?**.
5. If **Yes** were selected for the **Run Quicksim?** option, QuickSim would automatically be run on the design after it was prepared for timing simulation. In this case, you want to stop and analyze the timsim8.log file, so select **No**.
6. Press return or select **OK** to execute the command. The script produces a shell and runs in it.

Examining the timsim8.log File

Examine the timsim8.log file as follows:

1. In PLD_DMGR, select **Right Mouse Button → Update Window**. The window is updated with the files that TIMSIM8 generated.
2. Find the timsim8.log file and select it with the left mouse button.
3. Choose **Right Mouse Button → Open → Editor** to open the file in the editor. No errors or warnings should be reported. For a short summary of the commands executed by TIMSIM8 during the timing flow, see the ”Timing Simulation Command Summary” at the end of this chapter. The timing flow is always the same since the starting point is always a routed LCA file with delays.
4. When you have finished looking at the file, close the editor window.

Simulating with a Command File in QuickSim

1. Double-click on the QuickSim II icon in the tools window. A dialog box appears.

2. Type the appropriate Component name in the field labeled Design pathname. For example, use $xilinx_tutorial / calc_3ka / calc for the 3ka calc design.
3. Select the **Constraint** option for Timing mode.
4. Select the **Visible** option for Detail of ‘Constraint’ timing mode. A new set of buttons appears in the dialog box.
5. Select **Typ** for Timing mode. This specifies the use of the back-annotated timing information.
6. Select **Messages** for Constraint mode.
7. Leave the rest of the buttons set at their defaults, and press return to start QuickSim. For more information on these other options, refer to the Mentor Graphics documentation on QuickSim. For most Xilinx simulations, the above setup is appropriate.
8. Resize the QuickSim window so that it is as large as the entire screen.
9. At any location in the QuickSim window, type “dofile calc_3k.do” for 3k or 3ka designs, and “dofile calc_4k.do” for 4k or 4ka designs. This replays a transcript file similar to the one created earlier. This transcript file opens the design; opens Trace and Monitor windows with the correct signals; assigns stimulus to the signals; and then runs the simulation. It should be obvious when you look at the trace output that real delay values are being used. It may be useful to view the transcript file using the editor in PLD_DMGR or another editor.

Timing Simulation Command Summary

Although this tutorial uses PLD_FNCSIM8 and PLD_TIMSIM8 to process the Calc design, you do not have to use these scripts, although it is recommended. You can run any of the following programs from the system prompt. The commands are listed below as you would type them at the system prompt or in a script.

To run Men2XNF8 from the system prompt or a batch file, use:

men2xnf8 calc -p ***parttype***

To run FNCSIM8 from the system prompt or a batch file, type:

fncsim8 calc -o

The -o option specifies to use the original schematic, as opposed to generating a new, flat schematic using the -g option.

To create a timing simulation netlist with TIMSIM8, after implementing the design using PLD_XMake, type:

```
timsim8 calc -o
```

To run the individual programs to create a timing simulation netlist from a routed design, type:

```
lca2xnf -g calc calc_timaka
                            Translate LCA file to XNF netlist
unakaxnf calc_timaka
                            Remove aliases, only run if .aka file
                            exists (XC3000 only)
mv calc_timaka.xnf calc_tim.xnf   Rename file
xnfba calc.xff calc_tim.xnf -m
                            Restore original net names to XNF file
                            and produce Mentor Graphics
                            simulation .mbafile
pld_dve_ba calc_tim.mba
                            Annotate delay values to design
```

For more information on the Xilinx programs referenced in this tutorial, refer to the *XACT Reference Guide* or the *Mentor Graphics Version 8 Interface User Guide*.

For simulation command summaries for designs with X-BLOX or Xilinx ABEL modules, see the appropriate section in the applicable tutorial chapter in this manual.

Mentor Graphics Interface/ Tutorial Guide

X-BLOX Tutorial

Chapter 13

X-BLOX Tutorial

Introduction

The X-BLOX library contains module generators that describe a system using high-level functions instead of gate primitives. The X-BLOX synthesis tool processes these modules. Using X-BLOX you can reduce design entry time as well as create fast and efficient designs. The X-BLOX library can be used with XC3000A, XC3000L, XC3100A, and XC4000 FPGA designs.

This chapter provides a step-by-step example using X-BLOX in the Mentor Graphics design environment. It is not intended to provide a complete description of X-BLOX functionality. Please refer to the "Further Reading" section at the end of this tutorial for advice on where to go for more information.

Before Beginning the Tutorial

This tutorial assumes you are familiar with the Design Architect and QuickSim tutorial chapters. It is especially important that the MGC_LOCATION_MAP variable is set as described in the Design Architect or QuickSim tutorial chapters. If you are not familiar with these chapters, review them before continuing.

Required Software

The following versions of the development software are required to perform this tutorial:

- Mentor Graphics Version 8.2_5 or later
- Xilinx / Mentor Graphics Interface DS344 Version 5 or later

- XACT Design Manager (XDM) Version 5 or later
- X-BLOX DS380 Version 5 or later

Preparing the Design

You must have a completed Calc design to perform this tutorial. You can obtain this design by completing the Design Architect tutorial or by copying it from one of the tutorial solution directories. The tutorial files are optionally installed when you install the DS344 interface software. The tutorial directory contains the tutorial files. It is recommended that you copy the appropriate tutorial directory in its entirety from the original installation area to another area. Completed Calc designs are in the following directories:

- calc_3ka — solution files for XC3020APC68
- calc_4k — solution files for XC4003PC84 and XC4003APC84

Alternatively, if you completed the Design Architect tutorial, use the design you created if it is targeted for an XC4000 family or an XC3000A part.

Note: Do not use the calc_3k design since X-BLOX cannot be used to create XC3000 designs.

To copy a completed Calc design from a solution directory, perform the following steps:

1. Check for a tutorial / mentor directory under your DS344 directory. If necessary, install the tutorial files.
2. In the *DS344_install_path* / tutorial / mentor directory are two solutions directories, calc_3ka and calc_4k. Copy one of these directories using the copy command in PLD_DMGR. You must use PLD_DMGR to make a working copy of the tutorial.

Modifying the Calc Design

In this tutorial, you replace the ALU block in the Calc design with an ALU block created with X-BLOX. Because the ALU block performs many bus-oriented arithmetic logic functions it is suited for replacement with an X-BLOX module. The replacement block is called ALU_BLOX. ALU_BLOX is functionally equivalent to ALU, except that ALU_BLOX is implemented using X-BLOX components.

Adding X-BLOX Module to Calc

Replace the existing ALU block with the X-BLOX version as follows:

1. Start PLD_DMGR in the tutorial working directory, either calc_3ka, calc_4k, or calc_da.

Note: Make sure $MGC_WD is set to the working directory.

2. Open the top-level Calc schematic sheet in Design Architect.
3. Select only the ALU instance on the Calc schematic.
4. Select **`Right Mouse Button`** → **`Replace`** → **`Other.`** A dialog box appears.
5. Select the alu _blox design component from the dialog box navigator or type the component name in the text field. Press the return key.
6. The original ALU instance is replaced with ALU_BLOX. The name at the top of the symbol should now be ALU_BLOX.

Viewing the ALU_BLOX Schematic

View the ALU_BLOX schematic by pushing into the ALU_BLOX symbol as follows:

1. Select only the ALU_BLOX symbol on the Calc schematic.
2. Select **`File`** → **`Open Down`** from the menu bar and then open the schematic under the ALU_BLOX symbol. The schematic for ALU_BLOX appears similar to the one in the following figure. The DATA_REG module shown in the schematic is added in subsequent steps.

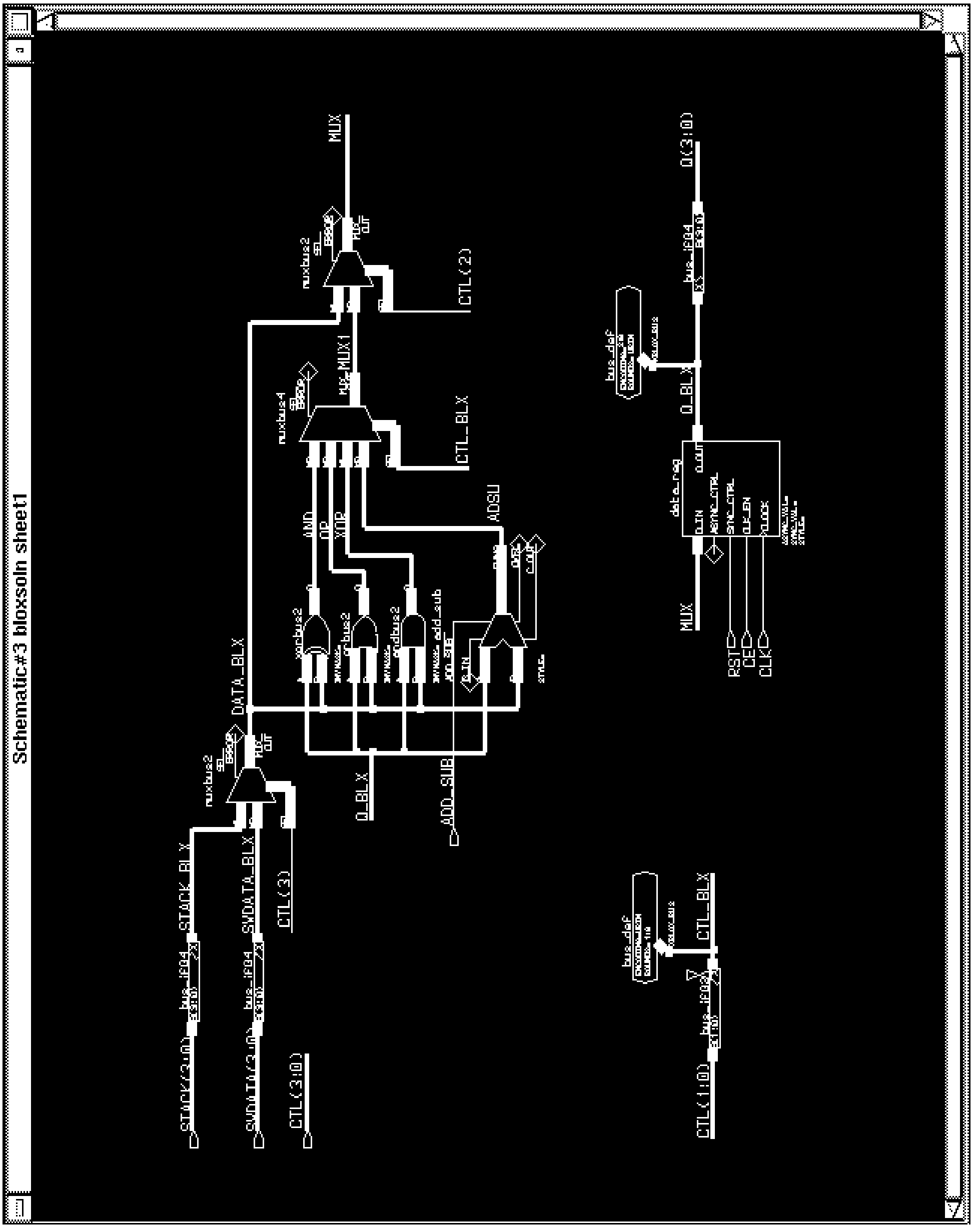


Figure 13-1 Completed ALU_BLOX Schematic

Completing the ALU_BLOX Schematic

Complete the ALU_BLOX using the schematic shown in the figure above and the following steps as a guide.

Note: The names of symbols and buses that you will add have been enlarged in the figure.

1. Open the X-BLOX Unified library, using the menu bar **`Libraries`** → **`XACT LIB`** option and the palette.
2. Find and select the DATA_REG library part from the X-BLOX library.
3. Place the DATA_REG symbol in the open area in the lower right-hand corner of the sheet, as shown in the figure above.
4. Connect and label the buses, MUX and Q_BLX, as shown in the ALU_BLOX schematic.
5. Connect and label the nets, RST, CE, and CLK, as shown in the ALU_BLOX schematic.
6. Select the BUS_IF04 component from the X-BLOX library.
7. Place the BUS_IF04 symbol to the right of DATA_REG as shown in the ALU_BLOX schematic.
8. Attach the bus Q_BLX to the end of the BUS_IF04 symbol labeled “X”.
9. Place a BUS_DEF symbol from the X-BLOX library above and between the DATA_REG and BUS_IF04 as shown in the ALU_BLOX schematic.
10. Attach the Q_BLX bus to the BUS_DEF.
11. Attach a bus to the BUS_IF04 pin B(3:0) and label it Q(3:0).

Understanding X-BLOX Buses

The bus pin output of the DATA_REG symbol has the name “Q”. In most cases, bus pins have names with an index, such as “Q(7:0)”, to specify the width of the bus pin. However, X-BLOX symbols are unique in that they do not have pre-defined bus widths. The same symbols are used, regardless of the size of the attached bus. Because of this difference, you must use a special interface to connect a non-indexed X-BLOX bus to a standard indexed bus. The BUS_IF symbols are used to interface an X-BLOX bus with a standard bus.

The bus interface used depends on the width of the bus that is interfaced. For example, in the ALU_BLOX schematic, the Design Architect bus STACK(3:0) is four bits wide and is interfaced to an X-BLOX bus with a BUS_IF04. CTL(1:0), a two bit bus, is interfaced with a BUS_IF02. You should connect only X-BLOX buses to X-BLOX symbol bus pins. You do not need an interface for individual nets that connect to X-BLOX symbols; for example, CTL2 and CTL3 on the two MUXBUS2 symbols.

Using BUS_DEF Symbols to Define Bus Widths

Attached to two buses in the schematic are BUS_DEFs, or bus definition symbols. By adding attributes to these symbols, you can define the properties of the entire data path attached to the BUS_DEF. Because you can define the properties of the entire data path and not just the properties of the bus directly connected to the BUS_DEF, the ALU_BLOX schematic requires only two BUS_DEF symbols. One BUS_DEF is needed for the four-bit data path through the ALU and another BUS_DEF is needed for the two-bit control signal path.

Use the BOUNDS attribute on a BUS_DEF symbol to define the width of the bus attached to the BUS_DEF. In the ALU_BLOX schematic, the BOUNDS value is 3:0 to indicate that the data path is four bits wide.

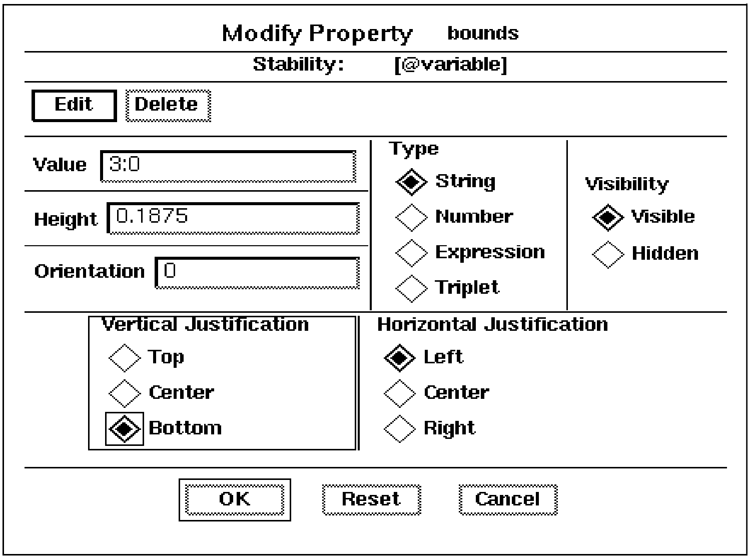
Use the ENCODING attribute to specify the type of data propagated on the data path. The possible choices are UBIN (unsigned binary), BIT (unsigned binary), TWO_COMP (two’s complement), or ONE_HOT (one-of-n). The type of ENCODING chosen can affect the functionality of every symbol on the data path. For example, the ADD_SUB block in the schematic is implemented as an unsigned binary adder/subtractor. If you specified TWO_COMP, the block would have been implemented as a two’s complement adder/subtractor.

Only certain ENCODING types are valid on a given data path. For example, you would not give the ADD_SUB data path a ONE_HOT encoding. However, you could use ONE_HOT encoding on the control path for the multiplexer. If the control lines that need to be attached to the mux are ONE_HOT, it would be necessary to define the ENCODING accordingly. In that case, choice of ENCODING completely alters the implementation of the mux.

Completing the Bus Definition

The definition of the ALU data path has not been specified. Add the following properties to the BUS_DEF symbol attached to the Q_BLX bus at the bottom of the ALU_BLOX sheet.

1. Select the BUS_DEF connected to the Q_BLX bus.
2. Select **`Properties → Modify.`** A dialog box appears containing a list of the properties attached to the symbol. Since the bounds and encoding properties must always be defined on a BUS_DEF, these two properties already exist on the symbol, but have undefined values.
3. Use the left mouse button to select **`bounds`** from the list of properties. Select **`OK`**. A new dialog box appears.
4. In the value field of the new dialog box, type **`3:0`** to give a value to the bounds property, as shown in the following figure.
5. Select **`OK`** to execute the command.
6. Repeat the necessary steps to assign the value, UBIN, to the ENCODING property.
7. Check and save the ALU_BLOX schematic.



Modify Property bounds

Stability: [@variable]

Value 3:0

Height 0.1875

Orientation 0

Type
◆ String
◇ Number
◇ Expression
◇ Triplet

Visibility
◆ Visible
◇ Hidden

Vertical Justification
◇ Top
◇ Center
◆ Bottom

Horizontal Justification
◆ Left
◇ Center
◇ Right

Figure 13-2 Modify Property Dialog Box

X-BLOX Symbols

The X-BLOX library contains elements that simplify designs by providing bus-oriented versions of logic, register, and multiplexing functions. You can place different attributes on X-BLOX symbols to customize them for specific applications. Also, X-BLOX implements macros differently depending on which pins are used on the symbol. This flexibility allows you to implement a wide range of different functions using the small set of parts in the X-BLOX library.

The following are examples of how attribute and pin usage affect the implementation of the X-BLOX macros in ALU_BLOX:

- DATA_REG

  The DATA_REG in this design has two attributes that can be set to alter its implementation, SYNC_VAL and ASYNC_VAL. These attributes are used to define the value that is loaded in the data register when it is synchronously or asynchronously reset. In this example, the data register should be reset to zero in either case, so both values are undefined, and default to zero. The SYNC_CTRL pin is connected, specifying a synchronous reset register.

- ADD_SUB

  The implementation of this ADD_SUB is affected by the definition of its data path and the pins connected to it. Since the data path is unsigned binary, the adder is implemented as an unsigned adder. The C_IN pin is not connected, and will default to the proper values for adding and subtracting. Since the ADD_SUB pin is connected, the ADD_SUB is implemented as an adder / subtractor.

- ANDBUS, ORBUS, MUXBUS*x*,...

  The implementation of the other X-BLOX symbols is similar to the original ALU design.The MUXBUS*x* symbols are affected by the ENCODING value of their attached buses.

  The bused logic symbols, such as ANDBUS and ORBUS, have a useful attribute, INVMASK, that affects their implementation. By changing INVMASK, you can invert the inputs to the symbol. For example, to invert input bit zero on the upper bus connection to the ANDBUS, select the ANDBUS and set the value for the INVMASK attribute to 2#0001#. The “1” in the string indicates the inversion of bit zero and the “2” indicates that the INVMASK value is specified in binary, with the total number of bits on the bus equal to four. All the INVMASKs in ALU_BLOX are undefined and default to a value of zero, indicating no bit inversions.

To illustrate another special characteristic of X-BLOX symbols, perform the following steps:

1. Select the DATA_REG instance on the ALU_BLOX schematic.
2. Attempt to open its underlying schematic using **`File → Open Down`** from the menu bar. A dialog box appears. Since X-BLOX macros do not have underlying schematic sheets, a schematic sheet does not appear in the dialog box. Select **`CANCEL`**.

Because X-BLOX macros adapt to any bus width and implement differently depending on data path encoding and pin usage, a single schematic cannot be used to represent the functionality of an X-BLOX macro. The schematic underneath the X-BLOX macros are created by the X-BLOX synthesis program, which is run by PLD_XMake and PLD_FNCSIM8. When you create a design using X-BLOX symbols, the information necessary for functional simulation does not exist. You must prepare the design for functional simulation as described in the next section.

Functional Simulation

A special set of programs allow you to easily simulate designs containing X-BLOX components. The execution of these programs has been encapsulated in the FNCSIM8 script. This script can be run using the PLD_FNCSIM8 icon in the Mentor Graphics Design Manager.

Note: For detailed information on FNCSIM8, please refer to the “Functional Simulation Preparation” chapter and the “QuickSim Tutorial” chapter.

To prepare an X-BLOX design for functional simulation you must run PLD_FNCSIM8. X-BLOX symbols are part of a set of objects that are referred to as non-schematic elements. This means that the functionality of the macro is not originally defined on the schematic, but is inserted in a later step. Because of this, some pre-processing of the schematic must be done to create the description of this missing functionality. PLD_FNCSIM8 runs the X-BLOX program, which reads X-BLOX macros and synthesizes the appropriate logic for them.

Once the functionality of the entire design has been described, the schematic itself must be modified. Recall that none of the X-BLOX buses or bus pins had defined bus widths on the original schematic. After the logic for the X-BLOX symbols is synthesized, these bus widths are available. PLD_FNCSIM8 runs a program called XBLXGS that redraws a new schematic. This new schematic is identical to the original schematic, except all of the buses and bus pins with unspecified widths are replaced by ones with specified widths. In addition, schematics are inserted underneath these modified symbols, so that the entire functionality of the design is then described on the schematic.

This new simulation-only schematic is placed in a directory called simdir, underneath the project directory. The simdir directory contains the modified design with the same name as the original. This is the design that is used for functional simulation.

Using PLD_Men2XNF8

Before you can functionally simulate your design, you must run PLD_Men2XNF8 to generate an XNF netlist as follows:

1. Quit Design Architect and enter PLD_DMGR.
2. Select the Calc design in the navigator window.
3. Select **Right Mouse Button → Open → pld_men2xnf8**. A dialog appears.
4. Type the appropriate part type in the Part Type field, 3020APC68, 4003APC84, or 4003PC84.
5. In not already selected, select **Use Original**.
6. Select **OK** or press return.
7. After the script finishes running, dismiss the window it was run in by pressing CTRL-C with the cursor in the window.

Men2XNF8 Log File

If errors occur, check the men2xnf8.log file for more information. A complete version of ALU_BLOX named BLOXSOLN is included in the solutions directories. If you cannot resolve the errors, replace ALU_BLOX with BLOXSOLN on the Calc schematic and then check and save. An example of a Men2XNF8 log file is shown in the following figure. The file is described below.

```
pld_dve /tutor/calc_3ka/calc xc3000

rm -f /tutor/calc_3ka/calc.edif

rm -f enwrite.cfg

/idea8.2/bin/enwrite /tutor/calc_3ka/calc/xnf -wef /tutor/
calc_3ka/calc.edif -rcf enwrite.cfg

rm -f /tutor/calc_3ka/calc.flt

rm -f /tutor/calc_3ka/calc.clb

edif2xnf /tutor/calc_3ka/calc.edif -l /ds344/data/unified/
edif3000 -lcanet 5 -n -p 3020APC68 -f -od /tutor/calc_3ka -of
calc.xnf
```

Figure 13-3 Men2XNF8 Log File

1.
   ```
   pld_dve /tutor/calc_3ka/calc xc3000
   ```

   PLD_DVE creates the XNF viewpoint for the design. The viewpoint defines which symbols are written to the netlist. For example, a two-input AND gate in the Calc design has a simulation model underneath it. In this case, the AND gate and not the underlying simulation model is written to the netlist. Conversely, the hierarchical block SW7 is not written directly to the netlist, but the symbols in the underlying schematic are represented in the netlist.

2.
   ```
   rm -f /tutor/calc_3ka/calc.edif
   rm -f enwrite.cfg
   /idea8.2/bin/enwrite /tutor/calc_3ka/calc/xnf -wef /tutor/
   calc_3ka/calc.edif -rcf enwrite.cfg
   ```

   In these steps, any existing design .edif file or enwrite.cfg configuration file is deleted to prepare for the generation of a new EDIF file by ENWRITE. ENWRITE is a Mentor Graphics program that produces an EDIF netlist from a Mentor Graphics design. It reads the XNF viewpoint created by PLD_DVE and produces an EDIF file that contains only Xilinx primitives.

3.
   ```
   rm -f /tutor/calc_3ka/calc.flt
   .
   .
   rm -f /tutor/calc_3ka/calc.clb
   ```

   PLD_Men2XNF8 creates a number of files that serve as flags to other scripts. For example, if a *design*.clb file exists in the design directory, this indicates the presence of XC3000 CLB primitives in the netlist. Before continuing, PLD_Men2XNF8 initializes the project directory by deleting any of these files that may be left in the directory from a previous run.

4.
   ```
   edif2xnf /tutor/calc_3ka/calc.edif -l /ds344/data/unified/
   edif3000 -lcanet 5 -n -p 3020APC68 -f -od /tutor/calc_3ka -of
   calc.xnf
   ```

   EDIF2XNF converts the EDIF file produced by ENWRITE to a standard Xilinx Netlist (XNF) file. One XNF file is produced for every block in the schematic.

Using PLD_FNCSIM8

Next, run PLD_FNCSIM8 to generate a schematic for functional simulation, using the information in the XNF file and the original schematic.

1. Select the Calc design in the navigator window.
2. Select **Right Mouse Button → Open → pld_fncsim8**. A dialog appears.
3. If not already selected, select **Use Original**.
4. Select **OK** or press return.

The FNCSIM8 script creates a shell and executes within it. FNCSIM8 flattens the XNF file created by Men2XNF8 and then checks it for errors. This netlist is then processed by the X-BLOX program. X-BLOX produces netlists that represent the functionality of the X-BLOX symbols in the design, and also creates a file that describes how to redraw the simulation schematic. The XBLXGS program then redraws the schematic using this information. The Gen_Sch8 program generates the simulation models for the X-BLOX symbols on the schematic from the XNF netlists. As noted earlier, this new schematic is placed in the simdir directory, underneath the project directory. The entire process of running PLD_FNCSIM8 may take several minutes.

FNCSIM8 Output

If FNCSIM8 returns errors, check the fncsim8.log, calc.prp, and calc.blx files for details. A complete version of ALU_BLOX named BLOXSOLN is included in the solutions directories. If you cannot resolve the errors, replace ALU_BLOX with BLOXSOLN on the Calc schematic and then check and save. The commands shown in the following figure are included in the FNCSIM8 output window; the entire output is not shown. The commands are described below.

Note: The complete output is not shown

```
xnfmerge /tutor/calc_3ka/calc.xnf /tutor/calc_3ka/calc.xff

xnfprep /tutor/calc_3ka/calc.xff /tutor/calc_3ka/calc.xtf

rm -r simdir

mkdir simdir
```

```
xblox /tutor/calc_3ka/calc.xtf simdir=simdir sim=xnf

xblxgs calc /tutor/calc_3ka/simdir/calc.xgs -w -d simdir

gen_sch8 simdir/bsm1.xnf -w -o simdir/bsm1

.

.

gen_sch8 simdir/bsm9.xnf -w -o simdir/bsm9

pld_dve_sim /tutor/calc_3ka/simdir/calc xc3000
```

Figure 13-4 FNCSIM8 Output

1. xnfmerge /tutor/calc_3ka/calc.xnf /tutor/calc_3ka/calc.xff

   XNFMerge takes the numerous XNF files produced by EDIF2XNF and merges them into one flattened netlist, which is given a .xff extension.

2. xnfprep /tutor/calc_3ka/calc.xff /tutor/calc_3ka/calc.xtf

   XNFPrep checks the flattened netlist for errors. Any errors are reported in a file with the same name as the design with a .prp extension. XNFPrep also trims unused logic from the netlist, and writes a new netlist with a .xtf extension.

3. rm -r simdir

   mkdir simdir

   In order to keep the new simulation schematic separate from the original design, it is generated in the new simdir directory. The simulation design has the same name as the original schematic, but is located in simdir.

4. xblox /tutor/calc_3ka/calc.xtf simdir=simdir sim=xnf

   X-BLOX is run on the flattened and trimmed netlist produced by XNFPrep. The X-BLOX program finds the X-BLOX symbols in the netlist, generates appropriate logic for them, and writes netlists describing this functionality to the simdir directory. It also produces a file with a .xgs extension that contains information describing how the simulation schematic should be drawn.

5. xblxgs calc /tutor/calc_3ka/simdir/calc.xgs -w -d simdir

   XBLXGS reads the .xgs file and the original schematic and produces a new top-level schematic in the simdir directory. In this schematic, all X-BLOX buses and bus pins are given indexed names.

6. gen_sch8 simdir/bsm1.xnf -w -o simdir/bsm1
   .
   .
   gen_sch8 simdir/bsm9.xnf -w -o simdir/bsm9

   The simulation schematic is completed by running Gen_Sch8 on the netlists produced by X-BLOX. This program creates simulation schematics for the X-BLOX symbols.

7. pld_dve_sim /tutor/calc_3ka/simdir/calc xc3000

   Finally, PLD_DVE_SIM is run to generate a simulation viewpoint for the new simulation schematic.

Viewing the Simulation Schematic

After FNCSIM8 is done, view the simulation schematic in Design Architect as follows:

1. In Design Manager, double-click on the simdir directory.
2. Select the Calc component in the simdir directory and open it in PLD_DA. To differentiate the simulation schematic from the original, all of the nets are green, otherwise, the schematic is identical to the original.
3. Open the schematic underneath the ALU_BLOX component. The following figure appears:

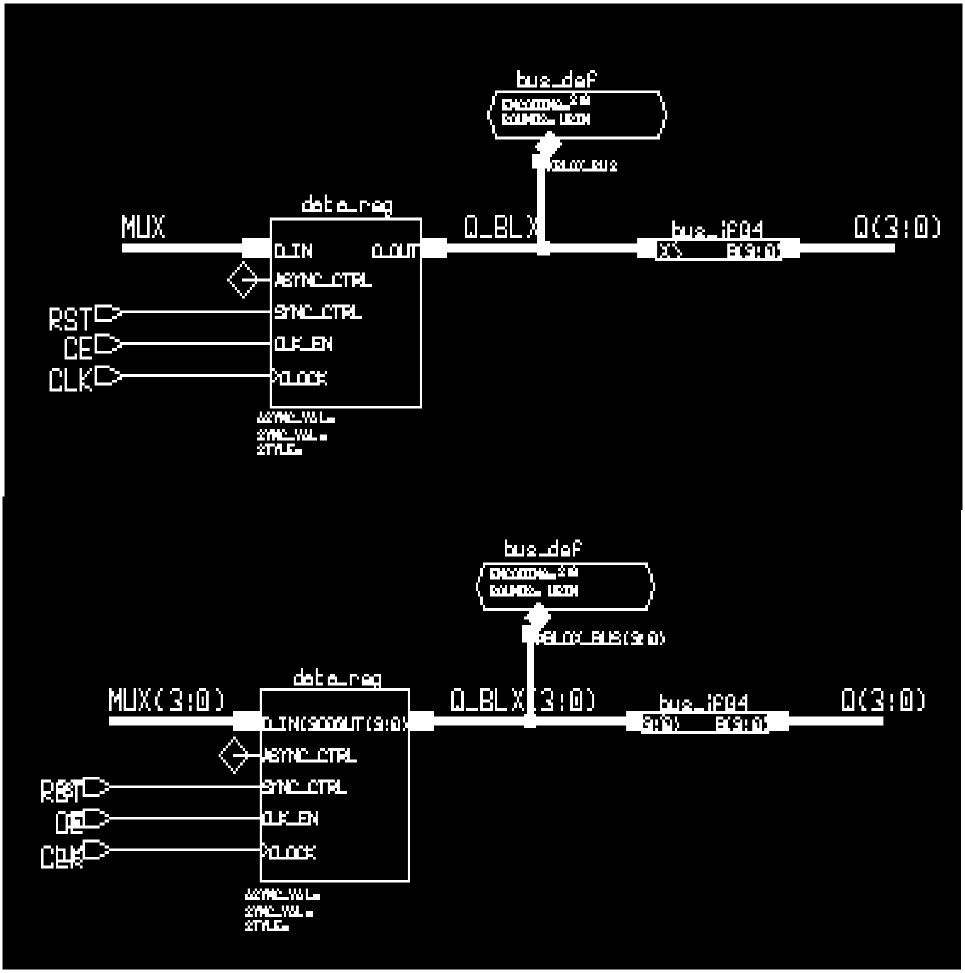


**Figure 13-5 (Top) DATA_REG from Original ALU_BLOX
(Bottom) DATA_REG from Simulation ALU_BLOX**

The X-BLOX symbols are replaced with their simulation equivalents. Note that the X-BLOX buses and bus pins have specific widths. Also, note the difference between the DATA_REG in the original ALU_BLOX in the Calc schematic and the one in ALU_BLOX from the simdir / calc schematic. The D_IN pin is now D_IN[3:0], and the OUT pin is OUT[7:0].

Note: You might have some difficulty noting this difference because the pin names overlap on the symbols when they are given indices.

If the X-BLOX symbols are not modified, be sure you have opened the Calc simulation schematic in the simdir directory and not the original Calc schematic. Inspect the fncsim8.log, men2xnf8.log, calc.prp, and calc.blx files for errors. Correct any errors and run FNCSIM8 again. Make sure that all attributes, nets, and buses you added to ALU_BLOX are spelled correctly. If you cannot resolve the problem, replace ALU_BLOX with BLOXSOLN.

Using QuickSim II

Open the simulation schematic in QuickSim as follows:

1. Close the Design Architect window.
2. In the Design Manager, open the simdir/calc design in QuickSim.
3. Resize the QuickSim window to cover the entire screen.
4. Execute either the calc_3k.do or calc_4k.do simulation command file, depending on whether you are working with an XC3000 or an XC4000 design, by typing **`dofile`** and one of the following path names:

Note: You must type the full path name since the simulation command file is not in the simdir directory.

- $xilinx_tutorial/calc_da/calc_3k.do or calc_4k.do
- $xilinx_tutorial/calc_3ka/calc_3k.do
- $xilinx_tutorial/calc_4k/calc_4k.do

The command file creates trace, list, and schematic windows; forces the inputs to values similar to those simulated in the QuickSim tutorial; and then runs the simulation for 3400 ns.

Note: The command file was created assuming a nineteen inch monitor. You may need to move or resize the windows, depending on the size of your monitor.

The output of the simulation should match the output from the functional simulation of the original Calc design without X-BLOX symbols. Refer to the “QuickSim Tutorial” chapter for more detailed information on the results of this simulation.

Implementing the Calc Design

Implementing designs that contain X-BLOX components is superficially identical to the translation of other designs. PLD_XMake can be used to generate files for timing simulation or bitstream for programming actual devices just as it is for non-X-BLOX designs. In the process XMake runs the X-BLOX program, which synthesizes the X-BLOX macros into standard logic gates.

For more information on PLD_XMake and the translation process, refer to the "Design Implementation" and the "Design Architect Tutorial" chapters as well as the *XACT Reference Guide*.

The design has not been modified since the last time we ran PLD_Men2XNF8, it is not necessary to run it again here.

Now, run PLD_XMake to generate files for use in timing simulation and device programming.

1. Quit QuickSim without saving and enter PLD_DMGR.
2. Select the Calc icon with the label XNF on top of it in the navigator window.
3. Select **Right Mouse Button** → **Open** → **pld_xmake**. A dialog box appears.
4. Keep the default settings and select **Done**.

PLD_XMake Output Window

The following output appears in the window created by PLD_XMake:

```
XMAKE: Generating makefile 'calc.mak'...
.
.

XMAKE: Making 'calc.bit'...

XMAKE: Execute command 'xnfmerge -D xnf -P 3020APC68-6
calc.xnf calc.xff'.

XMAKE: Execute command 'xnfprep calc.xff calc.xtg
parttype=3020APC68-7'.

XMAKE: Execute command 'xblox calc.xtg calc.xg
parttype=3020APC68-7'.

XMAKE: Execute command 'xnfprep calc.xg calc.xtf
```

```
parttype=3020APC68-7'.

XMAKE: Execute command 'xnfmap -P 3020APC68-7 calc.xtf
calc.map'.

XMAKE: Execute command 'ppr calc.map cstfile=calc_3ka.cst
parttype=3020APC68-7'.

XMAKE: Execute command ‘xdelay -D -W calc.lca’

XMAKE: Execute command 'makebits -R2 -S0 calc.lca'

XMAKE: 'calc.bit' has been made. Check output in 'calc.out'.
```

Figure 13-6 PLD_XMake Output Window

The following is a description of the steps in the output window of PLD_XMake:

1. XMAKE: Generating makefile 'calc.mak'...

   The PLD_XMake icon runs the XMake program. This is a core Xilinx program that processes XNF netlists to produce a placed and routed design. XMake always creates a makefile, in this case calc.mak, that allows you to stop the processing of a design at any point, and then continue processing later from the same point. This is accomplished by using the .mak file as input to XMake. Additionally, since this file contains a transcript of the programs run by XMake, it is useful as a starting point for the creation of customized processing scripts.

2. XMAKE: Execute command 'xnfmerge -D xnf -P 3020APC68-7 calc.xnf calc.xff'.

   XMAKE: Execute command 'xnfprep calc.xff calc.xtg parttype=3020APC68-7'.

   As in the functional simulation flow, XNFMerge combines the hierarchical XNF netlists created by EDIF2XNF into one large, flattened netlist, calc.xff. This netlist is then input to XNFPrep, which validates the contents of the file and performs logic trimming and optimization.

3. XMAKE: Execute command 'xblox calc.xtg calc.xg parttype=3020APC68-7'.

   Unlike the functional simulation flow, X-BLOX is run this time without the sim=xnf option. Since you are implementing the design, not simulating it, X-BLOX does not need to generate simulation models for every X-BLOX component in the design. Instead, it generates a single file, replacing the X-BLOX symbols in the netlist with synthesized logic.

4. XMAKE: Execute command 'xnfprep calc.xg calc.xtf parttype=3020APC68-7'.

   The new netlist produced by X-BLOX is again verified by XNFPrep, to ensure that no errors were introduced by X-BLOX.

5. XMAKE: Execute command 'xnfmap -P 3020APC68-7 calc.xtf calc.map'.

   For XC3000/XC3000A designs, XNFMap maps the logic to CLBs.

6. XMAKE: Execute command 'ppr calc.map cstfile=calc_3ka.cst parttype=3020APC68-7'.

   For XC3000A and all XC4000 family designs, PPR is run to place and route the design. PPR also performs the mapping step on XC4000 family designs. PPR reads the pin location constraints file specified by xdm.pro. In XC3000 designs, the APR program performs this step. The result of this step is the calc.lca file, which contains the place and route information.

7. XMAKE: Execute command ‘xdelay -D -W calc.lca’

   The XDelay program writes timing information to the calc.lca file, so that the timing information can then be back-annotated into simulation.

8. XMAKE: Execute command 'makebits -R2 -S0 calc.lca'
   XMAKE: 'calc.bit' has been made. Check output in 'calc.out'.

   MakeBits is run on calc.lca to produce a file you can use to program a device.This file is referred to as a bitstream, and is given the name calc.bit.

Verifying Calc on the Demonstration Board

At this point, a bitstream file has been created that can be downloaded to the appropriate demonstration board to verify the validity of the design. If you are unfamiliar with this process, please refer to the Design Architect tutorial or the XChecker section in the reference guide for more information.

Timing Simulation

When a design containing X-BLOX components is implemented, the X-BLOX symbols are expanded into blocks of logic. As a result, there is no longer a direct correlation between each symbol on the schematic and each block of logic in the LCA file. This creates a problem, because the delay information found in the LCA file must be back-annotated to a schematic before it can be used in simulation. Back-annotation is not possible without a direct correlation between the schematic and the LCA file. You must generate an entirely new schematic from the information found in the LCA file.

The schematic generated for functional schematic is based on the original schematic. The timing simulation schematic is based on information in the LCA file. The TIMSIM8 script creates a single page flat schematic from the information found in the LCA file.

Use TIMSIM8 to generate a schematic to use in timing simulation as follows:

1. Select the Calc component in the navigator window of PLD_DMGR.
2. Select **Right Mouse Button → Open → pld_timsim8**
3. Choose the **Auto Generate** radio button.

PLD_TIMSIM8 Output Window

TIMSIM8 executes and text similar to the following appears in the TIMSIM8 window:

```
lca2xnf -wg /tutor/calc_3ka/calc /tutor/calc_3ka/calc_timaka
/bin/mv -f /tutor/calc_3ka/calc_timaka.xnf /tutor/calc_3ka/
       calc_tim.xnf
xnfba /tutor/calc_3ka/calc.xg /tutor/calc_3ka/calc_tim.xnf -o /
       tutor/calc_3ka/calc_tim.xbf
mv -f /tutor/calc_3ka/calc_tim.xbf /tutor/calc_3ka/calc_tim.xnf
gen_sch8 -w /tutor/calc_3ka/calc_tim.xnf
```

Figure 13-7 PLD_TIMSIM8 Output Window

An explanation of the TIMSIM8 output follows:

```
lca2xnf -wg /tutor/calc_3ka/calc /tutor/calc_3ka/calc_timaka
```

LCA2XNF translates the calc.lca file that contains the timing information back into a standard XNF netlist, calc.xnf

```
/bin/mv -f /tutor/calc_3ka/calc_timaka.xnf /tutor/calc_3ka/
       calc_tim.xnf
```

The netlist output by LCA2XNF is changed to calc_tim. The timing simulation schematic eventually created from this file will also be named calc_tim.

```
xnfba /tutor/calc_3ka/calc.xg /tutor/calc_3ka/calc_tim.xnf -o /
       tutor/calc_3ka/calc_tim.xbf
```

In the process of placing and routing the design and then converting it back to an XNF netlist, some instance/net names and logic structures may no longer correspond to those found on the original schematic. The XNFBA program minimizes these differences by reading the original, pre-route XNF netlist and then rewriting the post-route XNF netlist, produced by LCA2XNF, so that the two are as similar as possible. This makes simulation easier since more net/instance names appear in simulation as they were originally entered. The output of XNFBA is given a .xbf extension.

```
mv -f /tutor/calc_3ka/calc_tim.xbf /tutor/calc_3ka/calc_tim.xnf
gen_sch8 -w /tutor/calc_3ka/calc_tim.xnf
```

The schematic generation tool Gen_Sch8, is run on the output of XNFBA to create a schematic you can use in simulation. The new schematic is given a "_tim" extension. Because there are X-BLOX symbols in the original schematic, it is not possible to use the original schematic for timing simulation. Instead, Gen_Sch8 produces a one-page, flat schematic. While this schematic is not very aesthetically pleasing, it is suitable for use in timing simulation since most design problems are resolved during functional simulation. An example of a portion of a timing simulation schematic produced from the Calc design by Gen_Sch8 is given in the figure below.

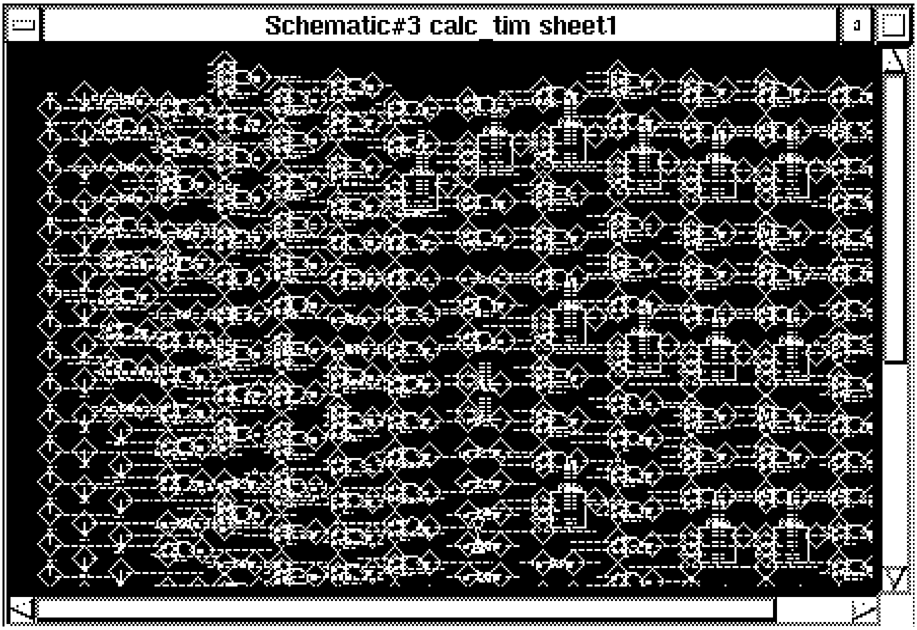


Figure 13-8 Portion of Calc Timing Simulation Schematic Produced by Gen_Sch8

Using QuickSim II

Enter QuickSim II and verify that the timing information has been back-annotated as described in the following steps:

1. Double-click on the QuickSim II icon in the Design Manager tools window. A dialog box appears.
2. In the field labeled Design pathname, enter the name of the Component created during the last step of the Gen_Sch8 program. For example, $xilinx_tutorial/calc_3ka/calc_tim for the 3ka calc design.
3. Select `Delay` for Timing mode.
4. Select `Visible` for Detail of 'Delay' timing mode. A new set of buttons appear in the dialog box.

5. Select the **Typ** for Timing mode. This specifies the use of the back-annotated timing information.
6. Select **Messages** for Constraint mode.
7. Leave the rest of the buttons set at their defaults, and press return to start QuickSim II. For more information on the QuickSim II options, refer to Mentor Graphics documentation. For most Xilinx simulations, the above selections are applicable.
8. Resize the QuickSim window so that it is as large as the entire screen.
9. Anywhere in the QuickSim window, type **dofile calc_3k.do** for 3K or 3KA designs, or **dofile calc_4k.do** for 4K or 4KA designs.
10. Check the trace output to confirm that real delay values are being used. You can view the transcript file using the editor in PLD_DMGR or using another editor.

Further Reading

Before beginning an X-BLOX design, read the descriptions of the X-BLOX macros found in the *XACT Libraries Guide* to understand the abilities and limitations of each macro. Also review the section on the X-BLOX executable program in the *XACT Reference Guide*. Additionally, see the *X-BLOX User Guide* for more detailed information.

Mentor Graphics Interface/ Tutorial Guide

Xilinx ABEL Tutorial

Printed in U.S.A.

Chapter 14

Xilinx ABEL Tutorial

Introduction

Xilinx ABEL allows you to define logic in terms of text-based Boolean equations, truth tables, and state machine descriptions using the ABEL Hardware Description Language (HDL). These logic blocks can then be included as part of a larger design, allowing logic defined by both graphical and text-based entry to exist in the same design.

This chapter provides a step-by-step example using Xilinx ABEL in the Mentor Graphics design environment. It is not intended to provide a complete description of Xilinx ABEL functionality. Refer to the “Further Reading” section at the end of this tutorial for information on additional Xilinx ABEL documentation.

Before Beginning the Tutorial

This tutorial assumes you are familiar with the Design Architect and QuickSim tutorial chapters. It is especially important that the MGC_LOCATION_MAP variable is set as described in the Design Architect or QuickSim tutorial chapters. If you are not familiar with these chapters, review them before continuing.

Required Software

The following versions of software are required to perform the tutorial:

- Mentor Graphics Version 8.2_5 or later
- Xilinx / Mentor Graphics Interface DS344 Version 5 or later
- XACT Design Manager (XDM) Version 5 or later

- Xilinx ABEL (DS371). This is the Xilinx text-based entry tool that uses the DATA I/O ABEL HDL language to enter logic descriptions.

Note: You can perform the tutorial without Xilinx ABEL licensing, since compiled netlists for the Xilinx ABEL code are provided.

To use Xilinx-ABEL, add the location of the DS371 Xilinx ABEL software to both the path and the XACT variable. See the DS371 release notes for more information.

Preparing the Design

You must have a completed Calc design to perform this tutorial. You can obtain this design by completing the Design Architect tutorial or by copying it from one of the tutorial solution directories. The tutorial files are optionally installed when you install the DS344 interface software. The calc_da directory contains the tutorial files. It is recommended that you copy the appropriate directory in its entirety from the original installation area to another area. Completed Calc designs are in the directories listed in the table below:

Table 14-1 Tutorial Design Directories

Directory	Description
calc_da	Tutorial Directory
calc_3k	Solution Directory for XC3020PC68
calc_3ka	Solution Directory for XC3020APC68
calc_4k	Solution Directory for XC4003PC84 and XC4003APC84

The solutions directories contain the design files for the completed tutorial, including schematics and the bitstream file. To conserve disk space, some intermediate files are not provided, except in the calc_3k directory, which is complete. Different intermediate files are created for different device families. Do not overwrite any files in the solutions directories.

Copying the Tutorial Files

Note: Do not use the Calc design in the calc_da directory unless you completed the Design Architect tutorial. If you did not complete the tutorial, use one of the solutions directories.

In PLD_DMGR, use the copy command to copy the applicable directory in its entirety to a different area. You must copy the directory in PLD_DMGR, so that the reference information is not corrupted. Do not use the UNIX cp command to copy a directory.

Modifying the Calc Design

An Xilinx ABEL-based block, STAT_ABL, is created in this section to replace the STATMACH (XC3000 family) or STAT_4k (XC4000 family) state machines that reside in the CONTROL block on the Calc schematic. Since the Xilinx ABEL code for STAT_ABL is functionally identical to the schematics for STATMACH and STATE_4k, this substitution does not alter the function of the Calc design.

Note: If you are not familiar with the Calc design, refer to the Design Architect tutorial.

Viewing STAT_ABL.ABL

STAT_ABL.ABL is the name of the Xilinx ABEL HDL file that is used to generate a logic description for the STAT_ABL block.

Enter the Xilinx ABEL environment and view the STAT_ABL.ABL source code by performing the following steps.

Note: If you do not have Xilinx ABEL, view the file with a text editor.

1. Enter the Xilinx Design Manager (XDM).
2. Select **`Design Entry → Xabel`**. A dialog box appears, allowing you to select an input .abl file.

Note: If Xabel does not appear in the menu, ensure that the directory containing it is referenced both in the path and in the XACT variable. Then, execute **`Utilities → ScanDisk`** in XDM to add it to the menu.

3. Select stat_abl.abl from the dialog box. XDM then runs Xilinx ABEL and loads the selected design into it.

Note: If stat_abl.abl does not appear in the dialog box, cancel the command and check to be sure you have the directory set correctly in XDM.

Text similar to the following appears in the main Xilinx ABEL window:

```
module stat_abl

title ‘State machine for Calc design’

“This state machine has 3 states which control the function

“of the ALU and the stack. The states are as follows:

“      SPUSH  -- increment stack pointer

“      SWE    -- write value into stack

“      SOTHER -- do neither (initial state)

“This is a one-hot state machine, which means that only

“one of the states is active at any given time. This method

“is particularly suited for use with Xilinx ABEL and Xilinx

“FPGAs, which are rich in flip-flop resources.

“This file also generates control signals from equations.

“For an equivalent schematic, see statmach.

declarations

“inputs

          OP5, OP4, OP3, OP2, OP1, OP0, EXC        pin;

“clock

          CLK                                      pin;

“outputs

          CTL3, CTL2, CTL1, CTL0                   pin;

          UP_DN, WE, RST, ADD_SUB, CE_ALU, CE_ADDR pin;

“state diagram declarations and assignments

          XABELSM        state_register istype 'reg_d';

          SPUSH, SWE, SOTHER                     state;

“vector definitions

          OP  = [OP5,OP4,OP3,OP2,OP1,OP0];

          HOP = [OP5,OP4,OP3];
          CTL = [CTL3,CTL2,CTL1,CTL0];
“declare internal nodes
          SEL_OP, OP_CTL2, OP_CTL1, OP_CTL0       node;
“node declarations for simulation only, can't use state names
“in simulation vectors
          PUSH, OTHER                             node;
“define clock & don't-care values for test vectors
          C, X = .C., .X.;
Xilinx property 'initialstate SOTHER';
equations
          XABELSM.CLK  = CLK;
          RST          = (HOP == ^b101) & EXC;
          ADD_SUB      = !OP_CTL2;
          SEL_OP       = (HOP == ^b111);
          CE_ALU       = !(SEL_OP & OP2 & OP0) & SOTHER & EXC;
          CE_ADDR      = !(OP2 & OP1 & OP0) & SEL_OP & EXC;
          OP_CTL2      = (OP5 & !SEL_OP) # (OP2 & SEL_OP);
          OP_CTL1      = (OP4 & !SEL_OP) # (OP1 & SEL_OP);
          OP_CTL0      = (OP3 & !SEL_OP) # (OP0 & SEL_OP);
          CTL3         = SEL_OP;
          CTL2         = OP_CTL2 & OP_CTL1;
          CTL1         = OP_CTL1 & !OP_CTL2;
          CTL0         = !OP_CTL2 & OP_CTL0;
          UP_DN        = OP2 & !OP1 & OP0 & SEL_OP & EXC;
          PUSH         = SPUSH;
          WE           = SWE;
          OTHER        = SOTHER;
“always optimize out don't-cares
@DCSET
state_diagram XABELSM
          state SPUSH:  goto                              SWE;
```

```
         state SWE:    goto                              SOTHER;
         state SOTHER: if                    (UP_DN) then SPUSH
                       else                              SOTHER;
test_vectors
“begin in initial state, each line is one clock cycle
([CLK,EXC,OP]-
>[PUSH,WE,OTHER,ADD_SUB,RST,CE_ALU,CE_ADDR,CTL])
“quick check to test the state machine
[ C, 0,   X ]->[  0,  0,  1,    X,     X,    X,     X,    X  ];
[ C, 1, ^h3F]->[  0,  0,  1,    X,     X,    X,     X,    X  ];
[ C, 0,   X ]->[  0,  0,  1,    X,     X,    X,     X,    X  ];
[ C, 1, ^h3D]->[  1,  0,  0,    X,     X,    X,     X,    X  ];
[ C, 0,   X ]->[  0,  1,  0,    X,     X,    X,     X,    X  ];
[ C, 0,   X ]->[  0,  0,  1,    X,     X,    X,     X,    X  ];
[ C, 1, ^h38]->[  0,  0,  1,    X,     X,    X,     X,    X  ];
“test the control logic, EXC low
[ C, 0, ^h0 ]->[  0,  0,  1,    1,     0,    0,     0,   ^h0 ];
[ C, 0, ^h8 ]->[  0,  0,  1,    1,     0,    0,     0,   ^h1 ];
[ C, 0, ^h10]->[  0,  0,  1,    1,     0,    0,     0,   ^h2 ];
[ C, 0, ^h18]->[  0,  0,  1,    1,     0,    0,     0,   ^h3 ];
[ C, 0, ^h20]->[  0,  0,  1,    0,     0,    0,     0,   ^h0 ];
[ C, 0, ^h28]->[  0,  0,  1,    0,     0,    0,     0,   ^h0 ];
[ C, 0, ^h30]->[  0,  0,  1,    0,     0,    0,     0,   ^h4 ];
“extended instruction set
[ C, 0, ^h38]->[  0,  0,  1,    1,     0,    0,     0,   ^h8 ];
[ C, 0, ^h39]->[  0,  0,  1,    1,     0,    0,     0,   ^h9 ];
[ C, 0, ^h3A]->[  0,  0,  1,    1,     0,    0,     0,   ^hA ];
[ C, 0, ^h3B]->[  0,  0,  1,    1,     0,    0,     0,   ^hB ];
[ C, 0, ^h3C]->[  0,  0,  1,    0,     0,    0,     0,   ^h8 ];
[ C, 0, ^h3D]->[  0,  0,  1,    0,     0,    0,     0,   ^h8 ];
[ C, 0, ^h3E]->[  0,  0,  1,    0,     0,    0,     0,   ^hC ];
[ C, 0, ^h3F]->[  0,  0,  1,    0,     0,    0,    0,   ^hC ];
“test the control logic, EXC high
[ C, 1, ^h0 ]->[  0,  0,  1,    1,     0,    1,    0,   ^h0 ];
[ C, 1, ^h8 ]->[  0,  0,  1,    1,     0,    1,    0,   ^h1 ];
[ C, 1, ^h10]->[  0,  0,  1,    1,     0,    1,    0,   ^h2 ];
[ C, 1, ^h18]->[  0,  0,  1,    1,     0,    1,    0,   ^h3 ];
[ C, 1, ^h20]->[  0,  0,  1,    0,     0,    1,    0,   ^h0 ];
[ C, 1, ^h28]->[  0,  0,  1,    0,     1,    1,    0,   ^h0 ];
[ C, 1, ^h30]->[  0,  0,  1,    0,     0,    1,    0,   ^h4 ];
“extended instruction set
[ C, 1, ^h38]->[  0,  0, 1,    1,      0,    1,    1,   ^h8 ];
[ C, 1, ^h39]->[  0,  0, 1,    1,      0,    1,    1,   ^h9 ];
[ C, 1, ^h3A]->[  0,  0, 1,    1,      0,    1,    1,   ^hA ];
[ C, 1, ^h3B]->[  0,  0, 1,    1,      0,    1,    1,   ^hB ];
[ C, 1, ^h3C]->[  0,  0, 1,    0,      0,    1,    1,   ^h8 ];
[ C, 1, ^h3D]->[  1,  0, 0,    0,      0,    0,    1,   ^h8 ];
“insert two clocks to return to initial state
[ C, 0, ^h3D]->[  0,  1, 0,    0,      0,    0,    0,   ^h8 ];
[ C, 0, ^h3D]->[  0,  0, 1,    0,      0,    0,    0,   ^h8 ];
[ C, 1, ^h3E]->[  0,  0, 1,    0,      0,    1,    1,   ^hC ];
[ C, 1, ^h3F]->[  0,  0, 1,    0,      0,    0,    0,   ^hC ];
end stat_abl
```

Figure 14-1 Xilinx ABEL Output

Xilinx ABEL Output

A description of the contents of the Xilinx ABEL file follows:

1. module stat_abl

   The module statement specifies the beginning of the Xilinx ABEL module.

2.
   ```
   title 'State machine for Calc design'
   ```

   The title statement is not necessary, but if used it is added as a header for the intermediate files created by Xilinx ABEL.

3.
   ```
   “This state machine has 3 states which control the functions
   “of the ALU and the stack. The states are as follows:
   “      SPUSH  -- increment stack pointer
   “      SWE    -- write value into stack
   “      SOTHER -- do neither (initial state)
   “This is a one-hot state machine, which means that only
   “one of the states is active at any given time. This method
   “is particularly suited for use with Xilinx ABEL and Xilinx
   “FPGAs, which are rich in flip-flop resources.
   “This file also generates control signals from equations.
   “For an equivalent schematic, see statmach.
   ```

   Any text preceded by a double quote, as in the above text, is interpreted as comment text.

4.
   ```
   declarations
   “inputs
   OP5, OP4, OP3, OP2, OP1, OP0, EXC       pin;
   “clock
   CLK                                 pin;
   “outputs
   CTL3, CTL2, CTL1, CTL0                  pin;
   UP_DN, WE, RST, ADD_SUB, CE_ALU, CE_ADDR pin;
   ```

   The pin statements in the declaration are used to define the pinout of the Xilinx ABEL module.

5.
   ```
   “state diagram declarations and assignments
   XABELSM       state_register istype 'reg_d';
   SPUSH, SWE, SOTHER                 state;
   ```

   The state_register keyword declares a symbolic state machine. The state keyword is used to declare states that appear in a symbolic state machine. “istype ‘reg_d’” declares that the state machine will be implemented using D flip-flops. State_register must be used in conjunction with state.

6.
   ```
   “vector definitions
   OP  = [OP5,OP4,OP3,OP2,OP1,OP0];
   HOP = [OP5,OP4,OP3];
   ```

```
CTL = [CTL3,CTL2,CTL1,CTL0];
```

Vector definitions are used to define bus vectors in Xilinx ABEL, which can be used during simulation in the Xilinx ABEL environment.

7.
   ```
   “declare internal nodes
   SEL_OP, OP_CTL2, OP_CTL1, OP_CTL0      node;
   ```

   The above nodes were declared for use as variables in intermediate equations.

8.
   ```
   “node declarations for simulation only, can't use state names
   “in simulation vectors
   PUSH, OTHER                            node;
   ```

   Since the Xilinx ABEL simulator does not allow the use of symbolic state names (those used in the definition of a state machine) in test vectors, two “dummy” nodes were created that mirror SPUSH and SOTHER, for use in the simulation test vectors found at the end of the file.

9.
   ```
   “define clock & don't-care values for test vectors
   C, X = .C., .X.;
   ```

   This definition allows for substitution of the default clock and don’t care syntax (.C and .X) with a simpler syntax without a period (C and X) to make the simulation vectors easier to read.

10.
    ```
    Xilinx property 'initialstate SOTHER';
    ```

    This defines the initial power-up state of the state machine as the state “SOTHER”. This state and others are defined in a later section of the file.

11.
    ```
    equations
    XABELSM.CLK  = CLK;
    RST          = (HOP == ^b101) & EXC;
    ADD_SUB      = !OP_CTL2;
    SEL_OP       = (HOP == ^b111);
    CE_ALU       = !(SEL_OP & OP2 & OP0) & SOTHER & EXC;
    CE_ADDR      = !(OP2 & OP1 & OP0) & SEL_OP & EXC;
    OP_CTL2      = (OP5 & !SEL_OP) # (OP2 & SEL_OP);
    OP_CTL1      = (OP4 & !SEL_OP) # (OP1 & SEL_OP);
    OP_CTL0      = (OP3 & !SEL_OP) # (OP0 & SEL_OP);
    CTL3         = SEL_OP;
    ```

```
CTL2        = OP_CTL2 & OP_CTL1;
CTL1        = OP_CTL1 & !OP_CTL2;
CTL0        = !OP_CTL2 & OP_CTL0;
UP_DN         = OP2 & !OP1 & OP0 & SEL_OP & EXC;
PUSH          = SPUSH;
WE          = SWE;
OTHER         = SOTHER;
```

The equation statement is used to define the internal logic of the module. Each equation statement is synthesized into combinatorial logic.

12.
    ```
    “always optimize out don't-cares
    @DCSET
    ```

    The @DCSET statement instructs Xilinx ABEL to optimize don’t-cares, as is done when you use Karnaugh maps to minimize a logic function.

13.
    ```
    state_diagram XABELSM
    state SPUSH:  goto                    SWE;
    state SWE:    goto                  SOTHER;
    state SOTHER: if              (UP_DN) then SPUSH
    else                  SOTHER;
    ```

    The state_diagram statement defines the circumstances that cause state transitions to occur. In this case, state SPUSH is always followed by SWE, SWE is always followed by SOTHER, and SOTHER is followed by SPUSH if the UP_DN signal is high. Otherwise, the state machine remains in the state SOTHER.

14.
    ```
    test_vectors
    ```

    The above line specifies the beginning of a section containing test vectors. The test vectors define sets of inputs and expected outputs.

15.
    ```
    “begin in initial state, each line is one clock cycle
    ([CLK,EXC,OP]
    [PUSH,WE,OTHER,ADD_SUB,RST,CE_ALU,CE_ADR,CTL])
    ```

    This line defines the set of inputs as the vectors CLK, EXC, and OP. The rest are outputs, for which expected values are specified below.

16. “quick check to test the state machine
“test the control logic, EXC low
“extended instruction set
“test the control logic, EXC high
“extended instruction set
“insert two clocks to return to initial state

Note: See the figure above for the complete output.

As noted above, each line represents one clock cycle, the inputs are specified in the square brackets to the left of the arrow, and the expected outputs are specified in the square brackets to the right of the arrow. The “^h” preceding some values indicates to the simulator that the vectors are specified in hexadecimal.

17. end stat_abl

The end statement specifies the end of the Xilinx ABEL module.

Verifying STAT_ABL

The Xilinx ABEL simulator is now used to verify the STAT_ABL design, using the test vectors described above as input.

Select **Compile → Simulate Equations.** A transcript window appears.

Xilinx ABEL prepares the test vectors for simulation and then simulates them. It should report that 39 of 39 test vectors simulated correctly. This means that as each of the test vector inputs was executed, the output of the state machine corresponded exactly to the expected values entered in the test vectors.

Note: If errors occur, it is possible that the Xilinx ABEL source code was inadvertently modified. Recopy stat_abl.abl from the appropriate solutions directory and try again.

Synthesizing STAT_ABL.ABL

Before replacing STATMACH with its Xilinx ABEL-based equivalent on the Mentor Graphics schematic, the Xilinx ABEL code must first be synthesized to a Xilinx Netlist Format (XNF) file, the standard file type for logic descriptions input to Xilinx tools. Also, a Mentor Graphics symbol must be generated.

While it is possible to generate an XNF file from stat_abl.abl in Xilinx ABEL, alternatively you will perform this task in XDM, using the ABL2XNF program.

If you do not have access to Xilinx ABEL or do not want to compile the code, the XNF file is provided in the solutions directories. If the steps below cannot be performed, rename the stat_abl.xnf.backup file to stat_abl.xnf, rename stat_abl.xsf.backup to stat_abl.xsf, and rename stat_abl.xas.backup to stat_abl.xas. You can then complete the tutorial by reading, but not performing the Xilinx-ABEL specific sections and completing only the remaining sections.

1. Select **File → Exit** in Xilinx ABEL and return to XDM.
2. In XDM, choose **Translate → Abl2xnf → stat_abl.abl**. An options dialog box appears.
3. Two very important options are the speed and area options. ABL2XNF has the ability to optimize code either for design speed or for device area. Choose the -speed option. The other files can be left at their defaults. Make sure only the -speed option is highlighted. If any other options are selected, deselect them using the left mouse button.
4. Choose **Done** to execute ABL2XNF.

ABL2XNF runs a number of translation and optimization tools and produces three important files: .xnf, .xas, and .xsf files. The .xnf file contains a partitioned netlist, for use in implementing the design. The .xsf file contains a symbol description that is used by the Gen_Sym8 program to create a Mentor Graphics symbol for the design. The .xas file contains an unpartitioned netlist, which is used for functionally simulating the design.

Under standard circumstances, it is not necessary to run any of the programs individually. Use ABL2XNF when possible to translate Xilinx ABEL designs.

Creating a Symbol for STAT_ABL

A special symbol must now be created so that the Xilinx ABEL module can be included in the CONTROL schematic. The Gen_Sym8 tool automates the creation of symbols for Xilinx ABEL modules. Gen_Sym8 uses as input the .xsf file created by Xilinx ABEL. This file contains the pinout for the symbol. Gen_Sym8 uses this file to generate an appropriate symbol

1. Gen_Sym8 is not available in XDM or PLD_DMGR, and must be run from a UNIX shell. Quit XDM and move to the shell where XDM was executed.
2. Change directories to the appropriate project directory (calc_da, calc_3k, calc_3ka, or calc_4k). Be sure the MGC_WD environment variable is set to the project directory.
3. Type **`ls *.xsf`**. The stat_abl.xsf file should be in the project directory. If it is not in the project directory, errors may have occurred during processing in Xilinx ABEL. Go back and try it again if the file does not exist.
4. Type **`gen_sym8 stat_abl.xsf`**. Gen_Sym8 executes, reads the .xsf file and writes out the symbol $MGC_WD / stat_abl.

Adding STAT_ABL to Calc

Now add the newly-created symbol to the Calc schematic.

1. Open PLD_DMGR.
2. Find the Control design component in the project directory and open it in PLD_DA. The schematic for CONTROL is displayed.
3. For 3K designs, select only the STATMACH instance on the CONTROL schematic. For 4K designs, select only the STATE_4K instance on the CONTROL schematic.
4. Select **Right Mouse Button → Replace → Other.** A dialog box appears.
5. Type the appropriate path to the newly-created STAT_ABL symbol in the Component Name field, or use the navigator in the dialog box to select it. For example, if you are working in the calc_3ka directory, the path would be $xilinx_tutorial / calc_3ka / stat_abl.

   This replaces the original STATMACH (or STATE_4k) block with the functionally equivalent Xilinx ABEL module, STAT_ABL. This change is reflected by the name STAT_ABL appearing at the top of the symbol.
6. Check and save the schematic.

The STAT_ABL symbol

The STAT_ABL symbol differs from the other symbols in the schematic because its logic is described in an XNF file instead of represented graphically using parts from the Xilinx libraries. To illustrate this, attempt to view the schematic for STAT_ABL by pushing into the STAT_ABL symbol as follows:

1. Select only the STAT_ABL symbol on the CONTROL schematic.
2. Select **`File → Open Down.`** A dialog box appears.

   Note that the only component that can be opened for STAT_ABL is the symbol. There is no existing schematic for it. The logic description for the STAT_ABL block is not defined inside Design Architect, but by the netlist stat_abl.xnf.
3. Press Escape to cancel the dialog box.

The FILE Property

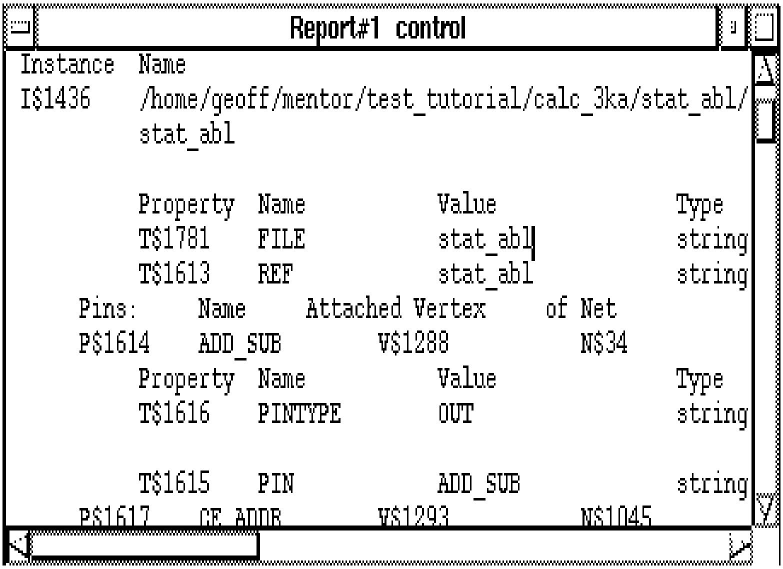
Since the logic description does not exist in the schematic, the tools that translate the design must be passed two important pieces of information:

- The logic description does not exist in an underlying schematic
- The logic description exists in a file

This information is specified by attaching a FILE property to the symbol. The value of the FILE property specifies the name of the file containing the logic description. In this example, the description is in the file STAT_ABL.XNF. This file was generated by Xilinx ABEL from the file containing the ABEL-HDL description of the logic.

Verify that the macro created by Gen_Sym8 placed the appropriate FILE property on the macro as follows:

1. Select only the STAT_ABL symbol on the CONTROL schematic .
2. Select **`Report→ Object → Selected`**. A report window appears. As shown in the following figure, the FILE property with the value, stat_abl, is attached to the symbol.
3. Close the report output window. Since you did not modify the schematic since the last save, you do not need to check or save it now.



```
Instance  Name
I$1436    /home/geoff/mentor/test_tutorial/calc_3ka/stat_abl/
          stat_abl

          Property  Name          Value             Type
          T$1781    FILE          stat_abl          string
          T$1613    REF           stat_abl          string
     Pins:     Name     Attached Vertex     of Net
     P$1614    ADD_SUB       V$1288          N$34
          Property  Name          Value             Type
          T$1616    PINTYPE       OUT               string

          T$1615    PIN           ADD_SUB           string
     P$1617    CE_ADDR       V$1293          N$1045
```

Figure 14-2 Results of Report on STAT_ABL Symbol

Functional Simulation

A special set of programs allow easy simulation of designs containing Xilinx ABEL components. The execution of these programs is encapsulated in the FNCSIM8 script. This script can be executed using the PLD_FNCSIM8 icon found in PLD_DMGR.

Note: For more information on PLD_FNCSIM8, refer to the “Functional Simulation Preparation” and “QuickSim Tutorial” chapters.

Use PLD_FNCSIM8 to prepare a Xilinx ABEL design for functional simulation. Unlike standard schematics, schematics containing Xilinx ABEL symbol blocks require extra processing. The simulation tools cannot directly use the information found in stat_abl.xnf, and a simulation schematic must be generated from the stat_abl.xnf file. This is done automatically by PLD_FNCSIM8, using the Gen_Sch8 program. Gen_Sch8 creates a new schematic that is inserted under the STAT_ABL symbol. Once this is done, the functionality of the design is described on the schematic, and it can be simulated.

PLD_Men2XNF8

Run PLD_Men2XNF8 to generate a netlist from the design as follows:

1. Quit Design Architect and re-enter PLD_DMGR.
2. Select the Calc design in the navigator window.
3. Select **Right Mouse Button → Open → pld_men2xnf8**. A dialog box appears.
4. Type the appropriate part type in the Part Type field.
5. Select **OK** or press return. The script creates a window and executes in it. Text similar to the text in the following figure appears.
6. Once the script completes, dismiss the window it executed within by pressing CTRL-C while holding the cursor in the window.

```
pld_dve /tutor/calc_3ka/calc xc3000
rm -f /tutor/calc_3ka/calc.edif
rm -f enwrite.cfg
/idea/bin/enwrite /tutor/calc_3ka/calc/xnf -wef /tutor/calc_3ka/
        calc.edif -rcf enwrite.cfg
rm -f /tutor/calc_3ka/calc.flt
.
.
rm -f /tutor/calc_3ka/calc.clb
edif2xnf /tutor/calc_3ka/calc.edif -l /ds344/data/unified/
        edif3000 -lcanet 5 -n -p 3020APC68 -f -od /tutor/calc_3ka
        -of calc.xnf
```

Figure 14-3 PLD_Men2XNF8 Output

PLD_Men2XNF8 Output

An explanation of the Men2XNF8 output follows.

```
pld_dve /tutor/calc_3ka/calc xc3000
```

PLD_DVE creates the XNF viewpoint for the design. This viewpoint defines which symbols are written to the netlist. For example, a two-input and gate in the Calc design has a simulation model underneath it. For the purposes of producing a netlist, the underlying simulation model is ignored and instead the and gate is written to the netlist.

Conversely, for the hierarchical block SW7, the symbols in the underlying schematic are represented in the netlist.

```
rm -f /tutor/calc_3ka/calc.edif
rm -f enwrite.cfg
/idea/bin/enwrite /tutor/calc_3ka/calc/xnf -wef /tutor/
       calc_3ka/calc.edif -rcf enwrite.cfg
```

In this step, any existing design .edif file or enwrite.cfg configuration file is deleted to prepare for the generation of a new EDIF file by ENWRITE. ENWRITE is a Mentor Graphics program that produces an EDIF netlist from a Mentor Graphics design. It reads the XNF viewpoint created by PLD_DVE and produces an EDIF file that contains only Xilinx primitives.

```
rm -f /tutor/calc_3ka/calc.flt
.
.
rm -f /tutor/calc_3ka/calc.clb
```

FNCSIM8 creates a number of files that serve as flags to other parts of the script, indicating such things as the presence of X-BLOX macros or FILE properties in a netlist. Before continuing, FNCSIM8 initializes the project directory by deleting any of these files that may be left from a previous run.

```
edif2xnf /tutor/calc_3ka/calc.edif -l /ds344/data/unified/
       edif3000 -lcanet 5 -n -p 3020APC68 -f -od /tutor/
       calc_3ka -of calc.xnf
```

EDIF2XNF converts the EDIF file produced by ENWRITE to a standard Xilinx Netlist (XNF) file. One XNF file is produced for every block in the schematic.

PLD_FNCSIM8

Run PLD_FNCSIM8 to generate a schematic that can be functionally simulated:

1. Select the Calc design in the navigator window.
2. Select **Right Mouse Button → Open → pld_fncsim8**. A dialog box appears.
3. Select **Use Original**.
4. **S**elect **OK** or press return.

At this point, the FNCSIM8 script creates a shell and executes within it. FNCSIM8 recognizes the presence of the FILE=stat_abl.xnf property in the netlist produced by Men2XNF8, and generates a model for STAT_ABL. A simulation viewpoint is created, and the design is ready for simulation.

Note: If FNCSIM8 returns errors, check the fncsim8.log for details.

The following commands appear in the FNCSIM8 window (most of the output is not shown, only the significant commands):

```
gen_sch8 stat_abl.xas -w -o stat_abl
pld_dve_sim /tutor/calc_3ka/calc xc3000
```

Figure 14-4 PLD_FNCSIM8 Output

PLD_FNCSIM8 Output

An explanation of the FNCSIM8 output follows:

```
gen_sch8 stat_abl.xas -w -o stat_abl
```

On finding the FILE property in the netlist, FNCSIM8 creates a simulation model for it. The simulation schematic is completed by running Gen_Sch8 on the stat_abl.xas netlist to produce a simulation schematic from the netlist.

```
pld_dve_sim /tutor/calc_3ka/simdir/calc xc3000
```

PLD_DVE_SIM is run to generate a simulation viewpoint for the new simulation schematic.

Viewing the Simulation Schematic

After FNCSIM8 completes, view the simulation schematic in Design Architect as follows:

1. In the Design Manager, select the appropriate control design component in the navigator window. For example, if you are using the XC3000A design, select $xilinx_tutorial/calc_3ka/control.
2. Choose **`Right Mouse Button → Open → pld_da.`** The control schematic appears in the Design Architect window.
3. Select the STAT_ABL component on the control schematic.

4. Open the schematic underneath the STAT_ABL symbol. Unlike previously, “schematic” appears as an option in the **`File`**→ **`Open Down`** dialog box, since Gen_Sch8 generated a simulation schematic for it.

5. The generated schematic appears, as shown in the figure below. This schematic is not very readable, but it is not intended to be used except for during timing simulation. Most design problems are resolved during functional simulation, so you will not need to reference the schematic very much during timing simulation. If the schematic does not appear, go back and check the fncsim8.log file for errors.

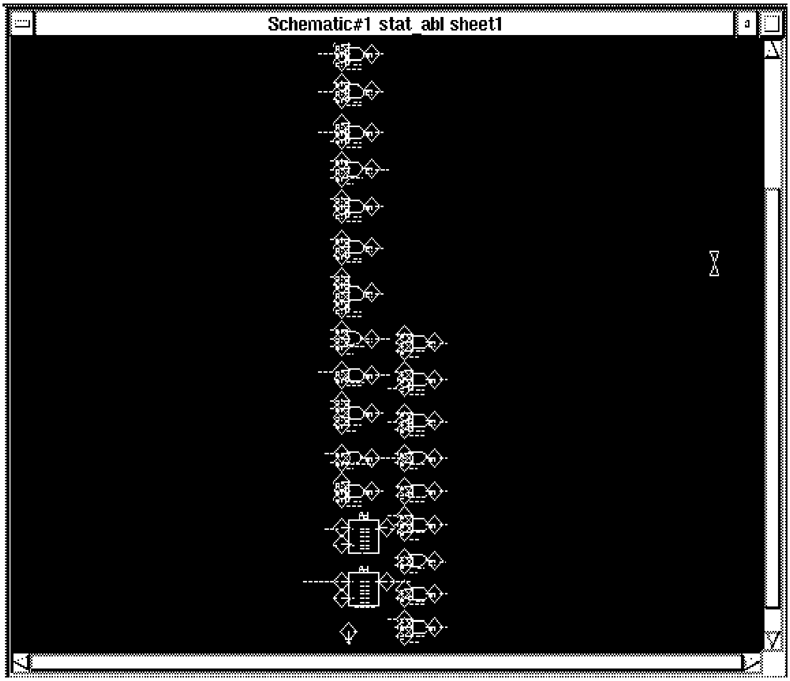


Figure 14-5 Portion of generated schematic for STAT_ABL

QuickSim II

Now that you have verified that the simulation schematic has been correctly created by FNCSIM8, open it in QuickSim and simulate as follows:

1. Close the Design Architect window.
2. In PLD_DMGR, open the Calc design in QuickSim.
3. Resize the QuickSim window so it covers the entire screen.
4. Execute either the calc_3k.do or calc_4k.do simulation command file (depending on whether you are using an XC3000 or an XC4000 design) by typing **`dofile`**, then the full path name to the command file (the command file path names are $xilinx_tutorial/ calc_3ka/calc_3k.do and $xilinx_tutorial/calc_4k/calc_4k.do).

Note: The command file creates trace, list, and schematic windows; forces the inputs to values similar to those simulated in the QuickSim tutorial; and then runs the simulation for 3400 ns. It may be necessary to move or resize the windows created by this script, depending on the size of your monitor. The command file was created assuming a nineteen inch monitor.

The output of this simulation run should be identical to the output of the functional simulation run on the original (non-Xilinx ABEL) Calc design. If the Design Architect tutorial was performed, the output from this simulation could be compared to the results saved in simrun1 (see Design Architect tutorial for more information). For a more detailed inspection of the results of this simulation of Calc, refer to the discussion found in the Quicksim tutorial.

Implementing the Calc Design

Implementing designs that contain Xilinx ABEL components is superficially identical to the translation of other types of designs. You can use PLD_XMake to generate timing simulation files or bitstream files for programming devices. If necessary, PLD_XMake runs the ABL2XNF program to synthesize the Xilinx ABEL file into a standard Xilinx netlist.

Note: For more information on PLD_XMake and the translation process, refer to the discussion of it found in the Design Architect tutorial. The *XACT Reference Guide* also contains information on the XMake program.

PLD_XMake

The design has not been modified since you ran PLD_Men2XNF8 to generate a netlist. This means you can skip the PLD_Men2XNF8 step, and can proceed directly to running PLD_XMake to generate files used in timing simulation and device programming.

1. Quit QuickSim, without saving and enter PLD_DMGR.
2. Select the calc.xnf file, produced earlier by PLD_Men2XNF8, in the navigator window. It will appear as an file icon with “XNF” written on it, and the word “calc” beneath it.
3. Select **`Right Mouse Button → Open → pld_xmake`**. A dialog box appears.
4. Select **`OK`**. Output similar to the following appears.

```
XMAKE: Generating makefile 'calc.mak'...
        .
        .
        .
XMAKE: Profile used is 'xdm.pro' file.
XMAKE: Makefile saved in 'calc.mak'.

XMAKE: Making 'calc.bit'...
XMAKE: Execute command 'xnfmerge -D xnf -P 3020APC68-6 calc.xnf
        calc.xff'.
XMAKE: Execute command 'xnfprep calc.xff calc.xtf
        parttype=3020APC68-6'.
XMAKE: Execute command 'xnfmap -P 3020APC68-6 calc.xtf
        calc.map'.
XMAKE: Execute command 'ppr calc.map cstfile=calc_3ka.cst
        parttype=3020APC68-6'.
XMAKE: Execute command 'makebits -R2 -S0 calc.lca'
        XMAKE: 'calc.bit' has been made. Check output in
        'calc.out'.
```

Figure 14-6 PLD_XMake Output

PLD_XMake Output

The following is a description of the PLD_XMake output:

```
XMAKE: Generating makefile 'calc.mak'...
```

The PLD_XMake icon runs the XMake program. This is a core Xilinx program that processes XNF files to produce a placed and routed design. XMake always creates a make file, in this case, calc.mak, that allows you to stop the processing of a design at any point, and then continue processing later from the same point. This is accomplished by giving the .mak file as input to XMake. Additionally, since this file contains a transcript of the programs run by XMake, it is useful as a starting point for the creation of customized processing scripts.

```
XMAKE: Profile used is 'xdm.pro' file.
```

XMake reads the xdm.pro file in the directory. This file specifies which options XMake should use when running programs. In this case, xdm.pro specifies the name of the appropriate pin location constraints file that is used by the place and route tools.

```
XMAKE: Execute command 'xnfmerge -D xnf -P 3020APC68-7 calc.xnf
        calc.xff'.
XMAKE: Execute command 'xnfprep calc.xff calc.xtf
        parttype=3020APC68-7'.
```

XNFMerge combines the hierarchical XNF netlists created by EDIF2XNF into one large, flattened netlist (calc.xff). This netlist is then input to XNFPrep, which validates the contents of the file and does logic trimming and optimization.

```
XMAKE: Execute command 'xnfmap -P 3020APC68-7 calc.xtf
        calc.map'.
```

For XC3000 / XC3000A designs, mapping (logic is broken into parcels that fit in CLBs) is performed by XNFMap. This step is performed by PPR for XC4000 family designs.

```
XMAKE: Execute command 'ppr calc.map cstfile=calc_3ka.cst
        parttype=3020APC68-7'.
```

For XC3000A and all XC4000 family designs, PPR is run to place and route the design (PPR also performs the mapping step on XC4000 family designs). The pin location constraints file specified by xdm.pro is read by PPR. (In XC3000 designs, the APR program performs this step). The result of this step is a file containing the place and route information, calc.lca.

```
XMAKE: Execute command 'makebits -R2 -S0 calc.lca'
       XMAKE: 'calc.bit' has been made. Check output in
       'calc.out'.
```

MakeBits is run on calc.lca to produce a file that can be used to program a device. This file is referred to as a bitstream, and is given the name calc.bit.

Verifying Calc on the Demonstration Board

At this point, a BIT file has been created that you can download to the appropriate demonstration board to verify the validity of the design. If unfamiliar with this process, please refer to the Design Architect tutorial for more information.

Timing Simulation

When a design containing Xilinx ABEL components is implemented, the logic for the Xilinx ABEL component on the schematic is merged into the netlist by XNFMerge. This means that there is logic in the placed and routed design.lca file that does not exist on the schematic. Because of this, there is no longer a direct correlation between each symbol on the schematic and each block of logic in the resulting LCA file. This creates a problem, because the delay information found in the LCA file must be back-annotated to a schematic before it can be used in simulation. Without a direct correlation between schematic and LCA file, this back-annotation is not possible. It then becomes necessary to generate an entirely new schematic from the information found in the LCA file.

PLD_TIMSIM8

Use PLD_TIMSIM8 to generate a schematic for use in timing simulation:

1. Select the Calc component in the navigator window of PLD_DMGR.
2. Select **Right Mouse Button → Open → pld_timsim8**
3. Choose **Auto Generate**.
4. PLD_TIMSIM8 executes.Text similar to the following appears in the TIMSIM8 window:

```
lca2xnf -wg /tutor/calc_3ka/calc /tutor/calc_3ka/calc_timaka
/bin/mv -f /tutor/calc_3ka/calc_timaka.xnf /tutor/calc_3ka/
       calc_tim.xnf
xnfba /tutor/calc_3ka/calc.xg /tutor/calc_3ka/calc_tim.xnf -o /
       tutor/calc_3ka/calc_tim.xbf
mv -f /tutor/calc_3ka/calc_tim.xbf /tutor/calc_3ka/calc_tim.xnf
gen_sch8 -w /tutor/calc_3ka/calc_tim.xnf
```

Figure 14-7 PLD_TIMSIM8 Output

PLD_TIMSIM8 Output

An explanation of the PLD_TIMSIM8 output follows:

```
lca2xnf -wg /tutor/calc_3ka/calc /tutor/calc_3ka/calc_timaka
```

LCA2XNF translates the calc.lca file, which contains the timing information, back into a standard XNF netlist, calc.xnf

```
/bin/mv -f /tutor/calc_3ka/calc_timaka.xnf /tutor/calc_3ka/
       calc_tim.xnf
```

Since the .xnf file is used later to produce a timing simulation schematic, it is given the name calc_tim.

```
xnfba /tutor/calc_3ka/calc.xg /tutor/calc_3ka/calc_tim.xnf -o /
       tutor/calc_3ka/calc_tim.xbf
```

In the process of placing and routing the design, and then converting it back to an XNF netlist, some instance and nets names and logic structures may no longer correspond to those found on the original schematic. XNFBA minimizes these differences by reading the original, pre-route XNF netlist and rewriting the post-route XNF netlist (produced by LCA2XNF) so that the two are as similar as possible. This simplifies simulation since more net/instance names appear in simulation as they were originally entered. The output of XNFBA is given a .xbf extension.

```
mv -f /tutor/calc_3ka/calc_tim.xbf /tutor/calc_3ka/calc_tim.xnf
gen_sch8 -w /tutor/calc_3ka/calc_tim.xnf
```

The schematic generation tool Gen_Sch8 is then run on the output of XNFBA, to create a schematic that can be used in simulation. The new schematic is given a “_tim” extension.

QuickSim II

Enter QuickSim and verify that the timing information has been back-annotated.

1. Double-click on the QuickSim II icon in the tools window. A dialog box appears.
2. Type the Component name **`calc_tim`**, preceded by the path to your working directory, in the Design pathname field. (for example, $xilinx_tutorial / calc_3ka / calc_tim for the 3ka calc design).
3. Select **`Delay`** for Timing mode.
4. Select **`Visible`** for Detail of ‘Constraint’ timing mode. A new set of buttons appears in the dialog box.
5. Select **`Typ`** for Timing mode. This specifies the use of the back-annotated timing information.
6. Select the **`Messages`** for Constraint mode.
7. Leave the rest of the buttons set at their defaults, and press return to start QuickSim. For more information on the options, refer to the Mentor Graphics documentation on QuickSim. For most Xilinx simulations, the above setup is appropriate.
8. Anywhere in the QuickSim window, type “dofile calc_3k.do” for 3k or 3ka designs, and “dofile calc_4k.do” for 4k or 4ka designs.
9. When you look at the trace output, it should be obvious that real delay values are being used. It may be useful to view the transcript file using the editor in PLD_DMGR or another editor.

Further Reading

Before beginning an Xilinx ABEL design, refer to the *Xilinx ABEL User Guide* and examine the design examples.

Mentor Graphics Interface/ Tutorial Guide

XACT-Performance and XDelay Tutorial

Printed in U.S.A.

Chapter 15

XACT-Performance and XDelay Tutorial

The specification of exact timing requirements on schematics has become a necessity as FPGAs have become larger and designs consequently more complex. XACT-Performance refers to the method used by the Xilinx software to describe these timing requirements. XACT-Performance consists of a set of library primitives that allow timing requirements to be placed on a schematic, along with built-in functionality in the PPR program that allows PPR to use this timing information during mapping, placement, and routing of the design.

XDelay is the companion tool that allows you to obtain exact timing information about the routed design created by PPR. Whenever you use XACT-Performance, verify the path timing using the XDelay program. To reduce run time, XACT-Performance does not use the highest possible level of accuracy in computing delays. XDelay reports completely accurate worst-case delays for all Xilinx FPGAs. Any differences between the two reports are minor, but when they occur, use the XDelay output as the definitive source for timing information.

Note: Since APR does not interpret XACT-Performance specifications, only XC3000A/L and all XC000 family designs can be used for this tutorial. XACT-Performance does not function on XC3000, XC3100, or XC2000 family designs.

The intent of this tutorial is to give you a practical example of using XACT-Performance and XDelay in the Mentor Graphics design environment. It is not intended to fully explain all of the functionality found in XACT-Performance or XDelay. Please refer to the “Further Reading” section at the end of this tutorial for a list of additional documentation you can refer to for more information.

Before Beginning the Tutorial

This section of the tutorial assumes that you are already familiar with the material in the “Design Architect Tutorial” and “Quicksim Tutorial” chapters of this manual. If not, please review those chapters before continuing.

Required Software

The following versions of the development software are required to perform this tutorial:

- Mentor Graphics 8.2_5 or later
- Xilinx / Mentor Graphics Interface — DS344 V5.00 or later
- XACT Design Manager — XDM V5.00 or later

Preparing the Design

1. The tutorial files are optionally installed when you install the DS344 interface software. If you have already installed the software but are not sure whether you specified tutorial installation, check for a directory named tutorial under your DS344 directory. The tutorial directory contains the tutorial files
2. You must copy a completed set of schematics and symbols from one of the solutions directories. Solutions for the tutorial are supplied in the following directories:

   calc_da — directory for Design Architect tutorial
   calc_3ka — solution files for XC3020APC68
   calc_4k — solution files for XC4003PC84 and XC4003APC84

   XACT-Performance cannot be used with XC3000 designs, so do not use the calc_3k design. It is recommended that you copy the appropriate tutorial directory in its entirety from the original install area to another area using the copy command available in PLD_DMGR. The copy directory must be given the same name as the original, since this directory name is encoded in the design component references. For more information on how to use PLD_DMGR to make a copy of a design directory, refer to the beginning of the Design Architect tutorial.

   The design in the calc_da directory should only be used if it was completed by performing the Design Architect tutorial. Otherwise, use one of the solutions directories.

Understanding XACT-Performance

When discussing the timing requirements of a design, it is simple to describe a requirement in such terms as “this signal must get from its source to this load in a certain amount of time.” XACT-Performance uses a similar from:to type of syntax. Symbols are grouped into classes, and these classes are then used as endpoints for timing specification. Timing requirements are defined as the maximum acceptable delays from the sources in one defined class, through intermediate combinatorial logic, to the associated loads in another class.

The three steps for adding timing specifications to a schematic are as follows:

1. Add TNM attributes to symbols on your schematic to group them into classes. This step is not necessary if you are using only predefined classes.
2. Add a TIMEGRP symbol and add properties to the symbol. These properties can combine the classes defined in step 1 into additional, more complex, classes. This step is optional.
3. Add a TIMESPEC symbol and add attributes to the symbol, defining the timing requirements for signals travelling between the classes defined in steps 1 and 2.

Grouping Symbols with TNM Attributes

The most basic and flexible way of defining classes is through the addition of TNM (Timing NaMe) attributes to symbols on a schematic. By giving two or more symbols TNM attributes with identical values, these symbols become part of the same class, which you can reference in a from:to statement.

TNMs on Logic Primitives

TNMs are applicable to four types of primitives: flip-flops, RAMs, I/O pads, and IOB latches. A class may not contain more than one of these types of symbols, with the exception of flip-flops and IOB latches, which may be included in the same class. TNMs on other primitives, such as OR gates, are invalid.

The syntax of the TNM attribute is as follows:

TNM=*identifier*

where *identifier* is replaced with the name of the class. The name can be any ASCII string using only the characters A-Z, a-z, _, and 0-9.

TNMs on Higher-Level Macro Symbols

You can also place TNM attributes on macro symbols containing one or more of the logic primitives just discussed. The TNM attribute is passed down through the hierarchy and placed on the logic primitives below. If the macro contains primitives of more than one type, you must specify the types of primitives inside the macro to which the TNM attribute applies. For example, a macro may contain RAMs and flip-flops. If you place a TNM on this macro, you must specify it as applying to either the RAMs or the flip-flops.

The syntax for applying TNM attributes to macros is as follows. You can specify one or more of the primitive types.

TNM=FFS:*identifier*;RAMS:*identifier*;LATCHES:*identifier*;
PADS:*identifier*

In this case, each instance of *identifier* is replaced by a unique class name, with the exception of FFS and LATCHES, which can be in the same class if desired.

TNMs on Nets to Tag Flip-Flops

The TNM attribute can also be placed on nets, using the TNM=*identifier* syntax. The software traces the signal to all load pins on the net and forward through combinatorial logic, and applies the TNM to any flip-flops reached. This spreading of TNM specifications to load pins is known as forward tracing.

For this purpose, if RAMs are encountered while tracing forward to load pins, they are seen as transparent. This means that if a flip-flop is sourced by the output of a RAM, and a TNM property is attached to the write enable of the RAM, the flip-flop becomes part of the class.

Grouping Symbols by Predefined Class

In many cases, it makes sense to apply a timing requirement to all associated symbols of a certain type. For example, a given flip-flop output may have a clock-to-setup timing requirement that applies to all other flip-flops driven by the flip-flop output.

To simplify the grouping procedure in such cases, Xilinx provides four predefined classes. These classes are FFS (flip-flops), PADS (I/O pads), RAMS (XC4000 family RAM elements), and LATCHES (IOB latches). Instead of placing a TNM attribute on each symbol, you can reference the entire class in a from:to statement by taking advantage of the predefined classes.

In the flip-flop example just discussed, you can use the from:to syntax to specify that the timing requirement be applied from the source flip-flop to the predefined class FFS.

Simplifying Symbol Grouping

The simplest way to group symbols is to use the basic syntax, TNM= *identifier*, on primitives. The other methods are shortcuts that enable you to quickly define classes that are related in some way, such as instances in the same macro, flip-flops driven by a common net, and so forth.

Combining Classes: TIMEGRP Symbol

Once classes are defined using TNM attributes, it can be useful to define new classes in terms of the existing classes. You may wish to join two or more classes into one, define a class that consists of all symbols not already included in another class, or designate a set of flip-flops triggered by a given clock edge as a new class. You can also use TIMEGRP to designate a class by the output net names of the primitive symbols.

To create these new classes, add the TIMEGRP symbol to your schematic, then add an attribute for each new class definition. The name of the attribute is the new class name. The value of the attribute is the class definition.

Each TIMEGRP symbol has room for eight class definitions. If you need to define more than eight classes, add additional TIMEGRP symbols to your schematic. You can place TIMEGRP symbols at any level of your hierarchy.

Joining Two or More Classes into One

You can define a new class as the combination of two or more existing classes using the following syntax:

***new_class*=*class1*:*class2*[...:*classn*]**

Using the EXCEPT Statement

A class defined using TNM attributes may account for all but a few of the flip-flops in a design. One way to apply timing specifications to the rest of the flip-flops is to create a new class that consists of all flip-flops not already in the first class. You can create the new class by defining an attribute that contains an EXCEPT statement. Use the following syntax:

new_class*=*class1*:EXCEPT:*class2

where *class1* is replaced by one of the predefined classes (FFS, PADS, RAMS, or LATCHES) or by the name of a user-defined class. *class2* is replaced by the name of a user-defined class.

For example, in the situation just discussed, assume that the class defined using TNMs is called FFGRP1, and the new class name is FFGRP2. You can create the class of all flip-flops not in the class FFGRP1 by adding the following attribute to the TIMEGRP symbol:

```
FFGRP2=FFS:EXCEPT:FFGRP1
```

Triggering on RISING or FALLING Clock Edges

You can also use TIMEGRP symbol attributes to make subsets of flip-flops that are triggered by a certain clock edge. Use the following syntax:

new_class*=RISING:*class

new_class*=FALLING:*class

New_class consists of all symbols in *class* that are clocked by the specified clock edge.

Forming Classes by Output Net Name

You can define a new class as the class of all primitives with output net names starting with a certain string. (BLKNMs or HBLKNMs are used for PADS, if you added these attributes; otherwise the external net name is used.) The full hierarchical net name is used.

You can also specify a class consisting of all blocks with output net names beginning with a certain name by using the following syntax:

new_class*=*class*(*name*)**

where *class* is one of FFS, RAMS, LATCHES, or PADS, and *name** limits the new class to those instances whose output net names contain the string *name*. The asterisk is a wildcard, and is not always necessary.

Note: This TIMEGRP capability must be used with caution. If your design contains unrelated nets with names beginning with the same string, they may be included in your time group or cause errors in XNFMerge or XNFPrep. If you attempt to apply the attribute to all blocks in a given schematic by specifying the instance name of the symbol, but the outputs of some flip-flops are renamed at a higher level of hierarchy, they are not included in the class.

Attaching Timing Specifications: TIMESPEC Symbol

Once you have defined appropriate classes by attaching TNM attributes to symbols, and, optionally, by combining these classes using the TIMEGRP symbol, the next step is to add the timing specifications to the schematic. First, place a TIMESPEC symbol on the schematic, then add the from:to timing requirements in the form of Mentor Graphics attributes. As with the TIMEGRP symbol, the TIMESPEC symbol itself has no electrical functionality, but serves as a placeholder for XACT-Performance attributes.

Use the following syntax to add timing specification attributes to a TIMESPEC symbol:

TS*name*=FROM:*class*:TO:*class*=*time*

All TIMESPEC attribute names must start with TS, followed by a unique identifier (*name* in the above example). The two *class* references are replaced with the appropriate class names, as defined by using TNMs and TIMEGRP symbols or by using the pre-defined classes. *Time* specifies the timing requirement, in microseconds (US), nanoseconds (NS), kilohertz (KHZ), or megahertz (MHZ). If no units are specified, *time* is assumed to be in nanoseconds.

For example, to specify that the pad to setup path delay between all pads and all flip-flops should be no greater than 40ns, add the following attribute to the TIMESPEC symbol:

```
TS01=FROM:PADS:TO:FFS=40NS
```

Note: FROM:PADS:TO:FFS is not exactly equivalent to pad to setup, since the PADS group includes not just data pads but also clock pads. Therefore, FROM:PADS:TO:FFS includes both pad-to-setup and pad-to-clock specifications. This inclusion is normally an advantage; however, if desired, you can use the EXCEPT syntax to eliminate the pad to clock paths. For example, to create a source group equivalent to the class referenced by pad to setup, use the TIMEGRP symbol to define a group such as PADS:EXCEPT:pads_sourcing_clocks.

Each TIMESPEC symbol has room for eight timing specification attributes. If you need more than eight specifications, add additional TIMESPEC symbols to your schematic. You can place TIMESPEC symbols at any level of your hierarchy.

Deciding When to Use XACT-Performance

The ideal approach to using XACT-Performance is to route your design once without any timing constraints. PPR by, default, controls path timing, using reasonable default values that it calculates based on your design. If the resulting LCA file meets your timing requirements, your design is complete.

If not, the PPR log file, ppr.log, gives values that can be achieved for FFS:TO:FFS, PADS:TO:FFS and FFS:TO:PADS timing. Use these values to help determine reasonable default timing requirements as described in the following section, “Setting Default Timing Requirements.”

After this run, check the new log file. If PPR is unable to meet the default timing for all paths, it reports the paths for which the default is not met. If critical paths in your design are not fast enough to meet your specifications, it is time to consider adding more specific constraints, as described in a later section “Adding Timing Constraints to Specific Paths”.

Clearly, tightening the default specifications for the entire design is unlikely to help PPR speed up the critical paths. Instead, consider a tighter specification on the most critical paths, combined with a looser specification for unimportant paths. You can even assign a value of IGNORE to some classes of paths, which is a very effective technique because it tells PPR that it can sacrifice timing on the unimportant paths to improve timing on the important ones. To use the IGNORE value, use the following syntax:

TS*name*=*path*:IGNORE

where *path* is one of the following special path types: DC2S, DC2P, DP2S, or DP2P.

You may want to skip the first step and start by setting reasonable default timing requirements.

If XACT-Performance and PPR are unable to achieve the speed needed for your application, you may have reached the limits of the hardware and/or software. You can increase the speed of the hardware by using a part with a faster speed grade. The tutorial designs are designed to work with the Xilinx Demonstration Boards and use the slowest available speed grades.

Consider speeding up your design by making changes to the logic to use the Xilinx FPGA architectures to better advantage. For example, try reducing the number of logic levels between flip-flops in critical paths. Xilinx FPGA architectures are rich in flip-flops, so pipelining is a good approach. Alternatively, you can often increase the speed of your design by using floorplanning. Use floorplanning to plan the placement of your logic to simplify the data flow and lock down symbol locations using CLBMAPs, FMAPs, HMAPs, and LOC attributes. See the *XACT Libraries Guide*, the *XACT Reference Guide*, and the other chapters of this manual for more information on how to floorplan your design.

Setting Default Timing Requirements

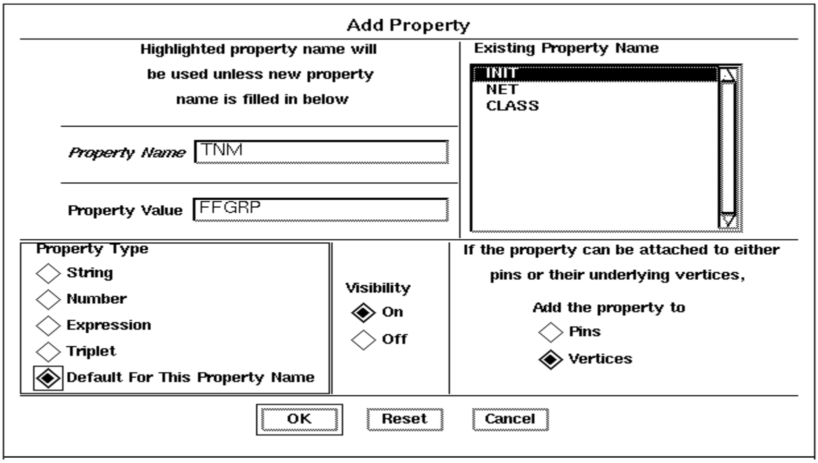
In this tutorial, you add XACT-Performance symbols and attributes to the Calc schematic but do not otherwise change the design. Many of the TIMESPEC constraints used in the this tutorial are not actually necessary for this or similar applications. They are included here to illustrate different uses of XACT-Performance that you may find useful in other designs.

Note: For more information on the Calc design, refer to the Design Architect tutorial.

Adding a TNM Property

First, use a TNM property to define a group of flip-flops. The group is used to specify the clock to pad default timing requirement.

1. Open Design Architect and load the Calc top-level schematic from the appropriate design directory (calc_da, calc_3k, calc_3ka, or calc_4k).
2. Making sure nothing else is selected, select a vertex (*not* an entire net segment) on the main clock net CLK and choose **`Right Mouse Button`** → **`Properties`** → **`Add`**. A dialog box appears.
3. Enter the data in the dialog box as shown in the figure below.



Add Property

Highlighted property name will be used unless new property name is filled in below

Property Name TNM

Property Value FFGRP

Existing Property Name
INIT
NET
CLASS

Property Type
◇ String
◇ Number
◇ Expression
◇ Triplet
◆ Default For This Property Name

Visibility
◆ On
◇ Off

If the property can be attached to either pins or their underlying vertices,

Add the property to
◇ Pins
◆ Vertices

Figure 15-1 Adding TNM Property

4. Select **`OK`**.
5. Place the property near the CLK net, using the left mouse button.

You have defined a class with the name FFGRP that consists of all flip-flops driven by the clock net CLK.

Note: Because there is only one clock in the Calc design, the FFGRP class in this case is the same as the predefined class FFS, which includes every flip-flop in the design. However, the class is defined to demonstrate the effects of attaching TNMs to clock signals, a very common technique in XACT-Performance.

Entering Default Timing Specifications

Next, set the default timing specifications for the clock. The clock in the XC3000A design should run at ~100Hz using the built-in on-board oscillator. The clock in the XC4000 design is driven by the on-chip oscillator, with the clock connected to the 15Hz output. We do not actually need to use XACT-Performance to meet these slow speeds. For simplicity, assume that in either case the clock speed is 500kHz. This clock speed is still quite slow, so the place and route software will meet these timing requirements easily.

1. Use the **`Add Component`** command to place a TIMESPEC symbol from the appropriate library (XC3000 for 3000A designs, XC4000 for 4000 designs) on the Calc schematic in the open area on the right.

Note: All of the XACT-Performance symbols can be found in the sub-library labeled “general” in the XACT_LIB library palette.

2. Making sure nothing else is selected, select the TIMESPEC symbol by clicking the left mouse button on it. The TIMESPEC symbol appears highlighted.
3. Choose **`Right Mouse Button → Properties → Add → Add Multiple Properties`**. A dialog box appears.
4. Add the following data in the property name and property value fields of the dialog box.

Property Name	Property Value
TS01	FROM:FFS:TO:FFS=500KHZ
TS02	FROM:PADS:TO:FFS=1MHZ
TS03	FROM:FFGRP:TO:PADS=1000 NS

5. Select **OK**, then place the properties on the TIMESPEC symbol.

Note: You may notice that the property names do not appear when the property is placed, only the property value is visible. This is a characteristic of the Mentor Graphics software. It may be desirable to place comment text next to the property values so that the name (TS01, TS02, and so on) associated with a property can be easily seen.

6. Check and save your changes to the Calc schematic.

The new attribute TS01 specifies that all clock-to-setup paths must have timing such that they can be driven by a 500kHz clock.

Pad-to-setup and clock-to-pad path delays are typically half of the clock-to-setup requirement. For the Calc design, they must be driven by a 1-MHz clock. TS02 specifies that all pad-to-setup path delays have timing such that they can be driven by a 1-MHz clock.

Making use of the FFGRP class, which in this case is equivalent to the FFS class, TS03 specifies that the clock-to-pad timing for the design be a maximum of 1000 ns. The two timing specifications of 1000NS and 1MHZ are interchangeable. An equivalent specification for TS03 is TS03=FROM:FFS:TO:PADS=1MHZ.

The following figure shows a portion of the Calc schematic with the TNM attribute and TIMESPEC symbol.

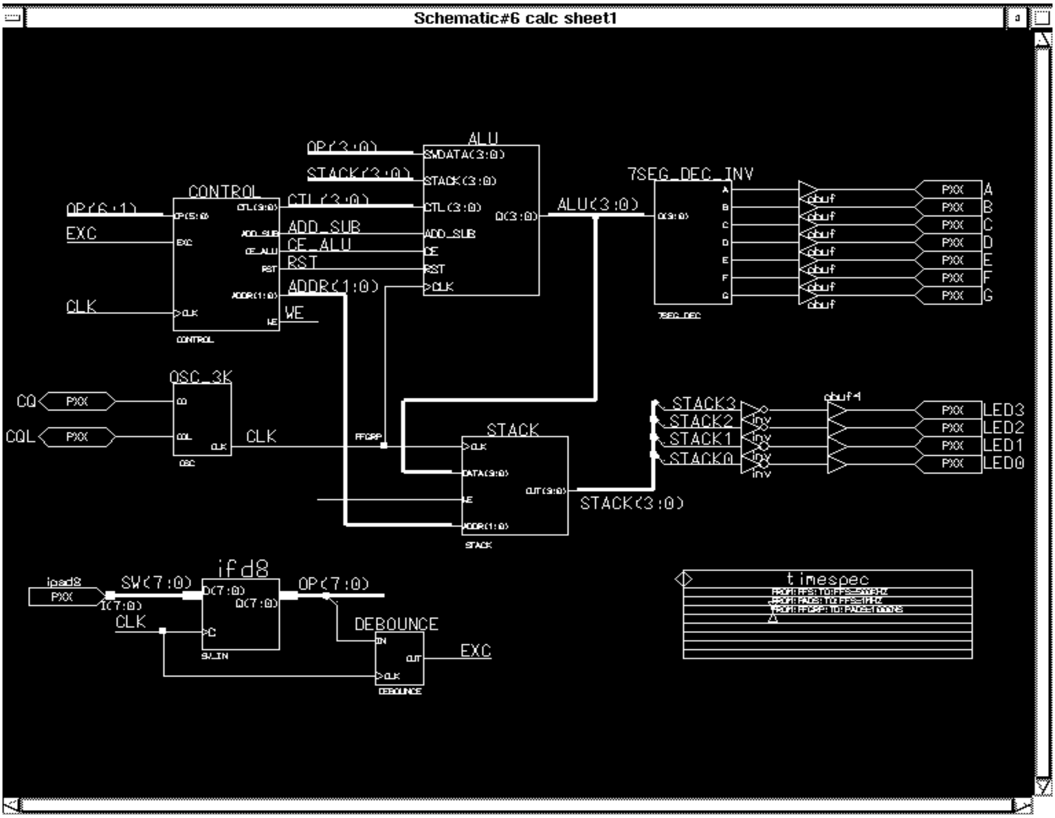


Figure 15-2 Calc Schematic with Default Timing Constraints

Adding Timing Constraints to Specific Paths

In the previous section, you applied default timing specifications to the Calc tutorial design. In real applications, the above specifications usually supply enough guidance to allow PPR to meet the timing requirements. However, in this section the tutorial continues with more specific path timing constraints to illustrate the application of XACT-Performance to more specific groups of paths in a design.

Defining TNM Groups

First, define classes to be used as endpoints for timing specification. You can use these classes to apply timing requirements from one class to another or from one class to the same class.

Defining the INFFS Class

The INFFS class includes all input flip-flops from the IFD8 macro.

1. Select the IFD8 symbol from the top level Calc schematic.
2. Choose **`Right Mouse Button`** → **`Properties`** → **`Add`**.
3. Enter the following data in the dialog box.

Property Name	Property Value
TNM	FFS:INFFS

4. Select **`OK`**, then place the property near or on the IFD8 symbol.

This property places the IFDs (IOB flip flops) in the IFD8 macro into a new timing class called INFFS. The keyword “FFS” specifies that the TNM property should only be applied to flip-flops underneath the selected symbol. If, for example, there were also pads or RAM underneath the symbol, these instances would not be placed in the TNM group. In this case, there are only flip-flops underneath IFD8, so the keyword is not absolutely necessary. However, if there is more than one symbol type underneath a macro and a keyword is not used to identify which symbol types the TNM is to apply, XMake will fail while running XNFMerge and issues a message explaining the error.

Defining the STACKER Class (XC4000 Family Only)

The Calc design for the XC4000 family contains a stack implemented using on-chip RAM elements. In this section, these RAM elements are grouped into a class called STACKER. If your design is an XC3000A design, skip this section and continue with the next section, “Defining the STACKER Class (XC3000A Only).”

1. Making sure nothing else is selected, select the STACK_4K symbol from the top level Calc schematic.
2. Choose **`Right Mouse Button`** → **`Properties`** → **`Add`**.

3. Enter the following data in the first row of the dialog box.

Property Name	Property Value
TNM	RAMS:STACKER

4. Select **OK**, then place the property on or near the STACK_4K symbol.

This attribute defines the new class named STACKER to contain all RAM symbols in the STACK _4K macro.

Note: The predefined class RAMS is optional in defining STACKER since the STACK_4K schematic contains only symbols of type RAM.

Defining the STACKER Class (XC3000A Only)

The Calc design for the XC3000A implements the stack using flip-flops. In this section, these flip-flops are grouped into a class called STACKER. If your design is an XC4000 family design, skip this section and continue with the next section, “Defining the ALUFF Class.”

1. Making sure nothing else is selected, select the STACK symbol from the top level Calc schematic.
2. Choose **Right Mouse Button → Properties → Add**.
3. Enter the following data in the first row of the dialog box.

Property Name	Property Value
TNM	FFS:STACKER

4. Select **OK**, then place the property on or near the STACK symbol

This attribute defines a new class named STACKER that contains all flip-flops in the STACK macro.

Note: The predefined class FFS is optional in defining STACKER since the only symbols in the STACK macro that can be given a TNM property are of type FFS.

Defining the ALUFF Class

1. Making sure nothing else is selected, select the ALU symbol from the top level Calc schematic.

2. Choose **Right Mouse Button → Properties → Add**.
3. Enter the following data in the first row of the dialog box.

Property Name	Property Value
TNM	ALUFF

4. Select **OK**, then place the property on or near the ALU.

This attribute defines a new class named ALUFF that contains all flip flops in the ALU macro. If a specific symbol type is not given (as was in the last two TNM definitions), the software assumes that the TNM should be applied to flip-flops. This is a useful shortcut if the only primitives pertaining to XACT-Performance underneath a macro are flip-flops (no rams or pads), as in the case of the ALU macro.

Defining the CTLFF Class

1. Making sure nothing else is selected, select the CONTROL symbol from the top-level Calc schematic.
2. Choose **Right Mouse Button → Properties → Add**.
3. Enter the following data in the first row of the dialog box.

Property Name	Property Value
TNM	CTLFF

4. Select **OK**, then place the property on or near the CONTROL symbol. This attribute defines a new class named CTLFF that contains all flip flops in the CONTROL macro.
5. Check and save your changes to the Calc schematic.

Defining the STFF Class

1. Verify that the CONTROL symbol is still selected.
2. Select **File → Open Down**. Open the schematic sheet for CONTROL.
3. Making sure nothing else is selected, select the STATE_4K symbol (XC4000 family designs) or the STATMACH symbol (XC3000A design).
4. Choose **Right Mouse Button → Properties → Add**.

5. Enter the following data in the first row of the dialog box.

Property Name	Property Value
TNM	STFF

6. Select **OK**, then place the property on or near the state machine symbol.

This attribute defines a new class named STFF that contains all flip-flops in the state machine. STFF is a sub-class of the CTLFF class. Even though this TNM attribute is defined on a lower-level schematic, it is valid to use it in the TIMESPEC symbol on a different level of the schematic.

Note: The above TNM attributes are all attached to macros, which simplifies grouping. Alternatively, you can attach TNM attributes to individual primitives, such as one or more individual flip-flops. In that case, attach the TNM=*class_name* attribute individually to each flip-flop you want included in the class.

Grouping Classes with TIMEGRP

In addition to the TNM classes, you can use the TIMEGRP primitive symbol to define new classes in terms of existing classes or predefined symbol types (FFS, RAMS, PADS, LATCHES).

Use the TIMEGRP symbol to create additional classes. Because the Calc schematic sheet is crowded, place the TIMEGRP symbol in the CONTROL schematic. TIMESPEC and TIMEGRP symbols can be placed at any level in the schematic.

1. Add a TIMEGRP symbol from the appropriate Xilinx library (3000 for 3000A designs, 4000 for 4000 family designs) to the CONTROL schematic in the open area at the lower right.
2. Make sure nothing is selected except for the TIMEGRP symbol.
3. Choose **Right Mouse Button → Properties → Add → Add Multiple Properties**.

4. Enter the following properties in the dialog box.

Property Name	Property Value
=LEDPADS	PADS(LED*)
=CTL_ALU_FF	CTLFF:ALUFF
=CTL_ADR_FF	CTLFF:EXCEPT:STFF

   Note that the property names start with an equal sign. The tools that write an XNF netlist from a design use a table that provides information on the Xilinx-specific properties that need to be written to the netlist, such as the LOC property. All other properties found in the schematic are not written to the netlist unless they are preceded by an equal sign. Since the class names on the TIMEGRP symbol are always different, there is no way to provide this information to the tools ahead of time. Because of this, the property names placed on the TIMEGRP symbol must be preceded by an equal sign.

5. Select **OK**, then use the left mouse button to place the properties on the TIMEGRP symbol.

The above procedure defines three new classes:

- The first new class, LEDPADS, represents all PADS symbols that begin with the character string “LED.” In this case, the pad symbols themselves do not have assigned BLKNM attributes, so XACT-Performance uses the full hierarchical names from the attached nets. The pads in the LED block are therefore named LED/LED0_P, LED/LED1_P, and so forth.
- The second class, CTL_ALU_FF, combines two TNM classes, CTLFF and ALUFF. This class includes all flip flops in either class.
- The third class, CTL_ADR_FF, represents all symbols in the CTLFF class except for those in the STFF class. Since CTLFF includes all flip-flops in the CONTROL block, and STFF includes all flip-flops in the state machine (STATE_4K or STATMACH), CTL_ADR_FF represents all flip flops in the CB2CLED macro below CONTROL.

6. Check and save your changes to the CONTROL schematic.
7. Close the CONTROL schematic and return to the Calc schematic.

The following figure shows the CONTROL schematic for the 3020A with the TNM on STATMACH and the TIMEGRP symbol.

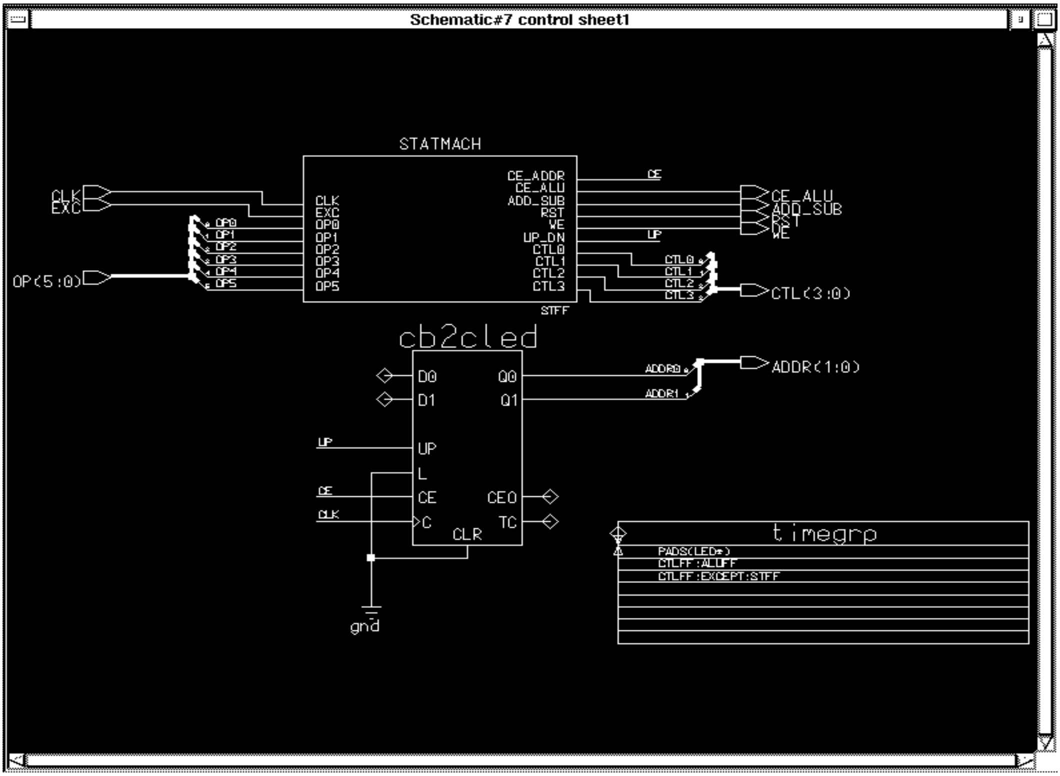


Figure 15-3 CONTROL Schematic with TNM and TIMEGRP

Specifying TIMESPEC Constraints

After completing all class definitions, specify the timing constraints. Use the defined classes from TNM and TIMEGRP and the predefined classes FFS, RAMS, PADS, and LATCHES as endpoints of the timing paths.

1. Select the TIMESPEC symbol by clicking the left mouse button on it.
2. Choose **`Right Mouse Button → Properties → Add → Add Multiple Properties`**.

3. Enter the following data in the dialog box.

Property Name	Property Value
TS04	FROM:INFFS:TO:FFS=80NS
TS05	FROM:CTL_ADR_FF:TO:ALUFF=50
TS06	FROM:CTL_ALU_FF:TO:STACKER=30
TS07	FROM:STACKER:TO:LEDPADS=50
TS08	FROM:ALUFF:TO:PADS=45

4. Select **OK**, then place the properties on the TIMESPEC symbol.
5. Check and save your changes to the Calc schematic.

The constraints that you have specified are the following.

- The TS04 timing attribute specifies that the clock-to-setup timing from the class of flip-flops named INFFS to all flip-flops (FFS) of the design be no more than 80 ns.
- TS05 specifies the clock-to-setup timing from the TIMEGRP CTL_ADR_FF to the class of flip-flops named ALUFF to be 50 ns.
- TS06 specifies the clock-to-setup timing from the TIMEGRP CTL_ALU_FF to the TNM class STACKER to be 30 ns.
- TS07 specifies the maximum path delays from the time the STACKER data becomes valid, plus any combinatorial delays, to the TIMEGRP LEDPADS, to be 50 ns.
- TS08 specifies the clock-to-pad timing from the TNM class ALUFF to all the pads in the design to be 45 ns.

Making a Final Check

Finally, check to make sure that the TIMEGRP and TIMESPEC symbols now have properties as shown in the following two figures. The order of the attributes is unimportant. Check this by first selecting the TIMEGRP symbol and executing **Right Mouse Button → Properties → Modify.** A dialog box appears showing the properties attached to the symbol. Check the properties, then dismiss the dialog box by pressing the Escape key. Repeat this for the TIMESPEC symbol as well.

TIMEGRP
=LEDPADS=PADS(LED*)
=CTL_ALU_FF=CTLFF:ALUFF
=CTL_ADR_FF=CTLFF:EXCEPT:STFF

Figure 15-4 Completed TIMEGRP Symbol

TIMESPEC
TS01=FROM:FFS:TO:FFS=500KHZ
TS02=FROM:PADS:TO:FFS=1MHZ
TS03=FROM:FFGRP:TO:PADS=1000NS
TS04=FROM:INFFS:TO:FFS=80NS
TS05=FROM:CTL_ADR_FF:TO:ALUFF=50
TS06=FROM:CTL_ALU_FF:TO:STACKER=30
TS07=FROM:STACKER:TO:LEDPADS=50
TS08=FROM:ALUFF:TO:PADS=45

Figure 15-5 Completed TIMESPEC Symbol

The completed Calc schematic is shown in the following figure. All desired XACT-Performance specifications have been entered on the schematic. The next step is to implement the design using XMake and verify the results in the calc.out and ppr.log files.

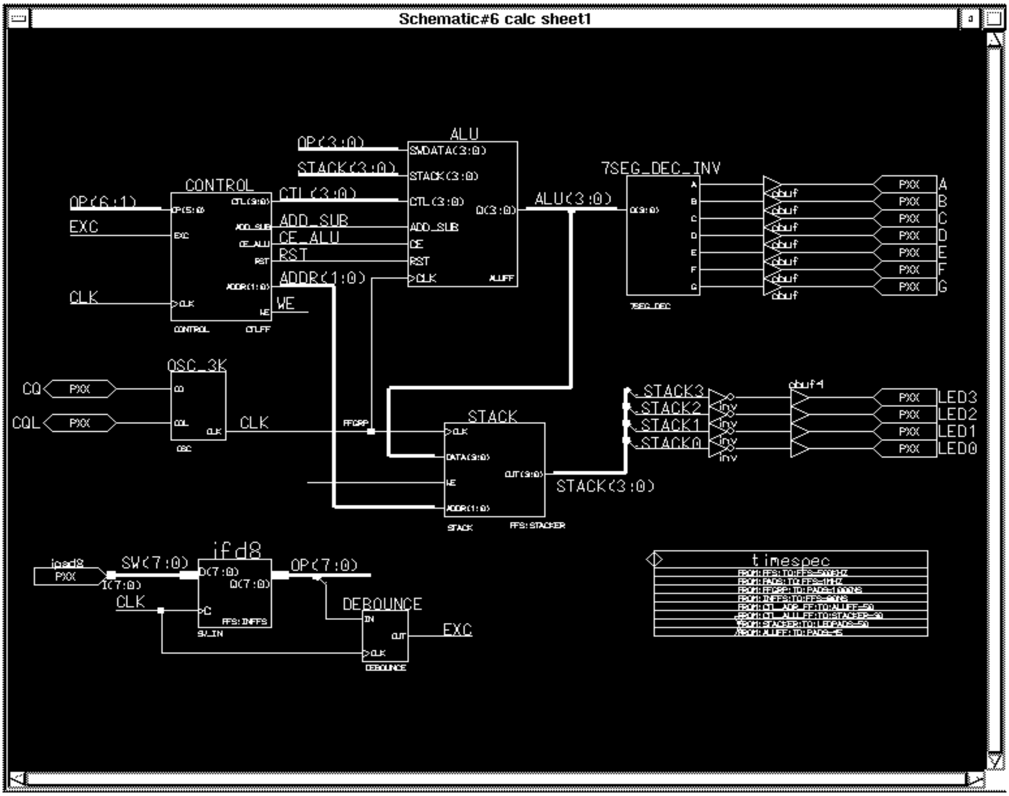


Figure 15-6 Calc Schematic with TNMs and TIMESPEC Symbol

Implementing the Calc Design

The translation of designs containing XACT-Performance attributes is exactly the same as the translation of other designs. In fact, even if you do not specify any XACT-Performance attributes, PPR by default controls path timing. PPR assigns reasonable default values and attempts to meet the imposed requirements.

If you apply XACT-Performance attributes to your schematics, PPR detects these specifications and, wherever they apply, uses them instead of calculating default values. In each phase of the implementation, which includes mapping, placement, and routing, PPR takes the XACT-Performance attributes into account. If PPR is unable to meet a given specification, it issues a warning to the PPR log file, relaxes the requirement, and continues.

Note: You can make PPR terminate when it encounters an XACT-Performance specification it cannot meet, by setting the PPR option stop_on_miss=true↵.

For more information on PPR, see the PPR section of the *XACT Reference Guide*. For detailed information on XMake and the translation process, refer to the “Design Architect Tutorial” chapter and to the *XACT Reference Guide*.

Creating a Routed Design

Run XMake to generate a routed LCA file.

1. Quit PLD_DA and open PLD_DMGR.
2. Select the Calc design component from the appropriate directory in the navigator window.
3. Select **`Right Mouse Button → Open → pld_men2xnf8`**.
4. Enter the appropriate part type and speed grade in the Part Type field, 3020PC68-70, 3020APC68-7, 4003PC84-6, or 4003APC84-6.
5. Using the default options, choose **`OK`**. The Men2XNF8 program is run to generate a Xilinx netlist (XNF) file from the schematic.
6. After Men2XNF8 executes, select the top-level calc.xnf file. It appears as a file icon named calc with the word “XNF” written on it.
7. Select **`Right Mouse Button → Open → pld_xmake.`**
8. Using the default options, choose **`OK`**. XMake maps, places, and routes the design.

Examining XMake Output

XMake produces a screen output similar to the following.

```
XMAKE: Generating makefile 'calc.mak' ...
XMAKE: Set the part type to '3020APC68-7 from 'calc.xnf'.
XMAKE: Profile used is the current XDM settings.
   >>> XDELAY is run always with '-D' and '-W' options by
XMAKE.
XMAKE: Makefile saved in 'calc.mak'.
XMAKE: Making 'calc.bit' ...
XMAKE: Execute command 'xnfmerge -D xnf -D . -P 3020APC68-7
calc.xnf calc.xff'.
XMAKE: Execute command 'xnfprep calc.xff calc.xtf
parttype=3020APC68-7'.
XMAKE: Execute command 'xnfmap -P 3020PC68-7 calc.xtf
calc.map'.
XMAKE: Execute command 'ppr calc.map parttype=3020APC68-7'.
XMAKE: Execute command 'xdelay -D -W calc.lca'
XMAKE: Execute command 'makebits -R2 -S0 calc.lca'
XMAKE: 'calc.bit' has been made. Check output in 'calc.out'
```

Figure 15-7 XMake Output

If your XACT-Performance specifications have any syntax errors, they are flagged by XNFPrep. XNFPrep is a tool that performs a series of checks on the XFF file, looks for illegal conditions of any sort, and then trims unused logic from the netlist. If XNFPrep detects an illegal XACT-Performance specification, XMake terminates. The XMake output as recorded in the calc.out file prompts you to check the calc.prp file for errors detected by XNFPrep. The PRP file includes a list of all errors and warnings issued by XNFPrep. In this case, the error message displays the attribute containing the syntax error and shows the correct syntax for the attribute.

If XNFPrep detects syntax errors in your design, locate the errors by checking the calc.prp file, correct the errors in the schematics, and run XMake again.

Note: Always check the *design*.out file after running XMake.

Examining the PPR Log File

After XMake completes, view the ppr.log file produced by PPR.

The figure below shows portions of the ppr.log file from the calc_3ka solutions directory design. The timing numbers reported vary depending on the target device. Additionally, unless you specify identical input parameters, each PPR run produces a slightly different result.

Note: Once you have a routed design that meets your timing needs, you can make changes to your design while retaining the timing characteristics of the unmodified logic. Use the incremental design procedure discussed in the “Making Incremental Design Changes” section of the Design Architect tutorial.

```
ppr 5.0.0 -- Xilinx Automatic CAE Tools
Copyright (c) 1994 Xilinx Inc. All Rights Reserved.

ppr: Reading input design data...
ppr: Placing logic...

*** PPR: WARNING 6811:

This design has 1 purely combinational loop. Such loops
should be avoided. If at all possible, please modify the
design to eliminate all unclocked feedback paths.

A loop of 1 source-to-load connections:
FG4 FG_OSC_3K/Q (Net OSC_3K/Q) to first gate again.

TS05: FROM_TO 50.0 ns;/ [best possible, 51.0 ns] = 0.98

*** PPR: WARNING 7015:
At least 1 of 5 path endpoints will miss spec. Continuing
anyway ..

ppr: Routing signals...
Design has 0 unroutes.

ppr: Generating .LCA File...
*** PPR: WARNING 10604:
An lca file already exists. The old lca file will be saved as
‘calc.lcb’.

ppr: Routing signals...

TS05: FROM_TO 50.0 ns;/ [best possible, 64.4 ns] = 0.78

*** PPR: WARNING 7015:
At least 4 of 5 path endpoints will miss spec. Continuing
anyway ..

Design has 0 unroutes.

------------------------------------------------------------
Timing analysis summary
------------------------------------------------------------

                     Deadline Actual(*) label: [qualifier]
                     -------- --------- ------------------

     ‘from’ ‘to’     1000.0ns   48.6ns     TSO3
     ‘from’ ‘to’     45.0ns     40.1ns     TS08:
```

```
    ‘from’ ‘to’      50.0ns      43.6ns      TS07:
    ‘from’ ‘to’      30.0ns      24.4ns      TS06:
(*) ‘from’ ‘to’      50.0ns      79.2ns      TS05:
    ‘from’ ‘to’      80.0ns      66.3ns      TS04:
    ‘from’ ‘to’      1000.0ns    29.4ns      TS02:
    ‘from’ ‘to’      2000.0ns    79.2ns      TS01:

(*) Note: the actual path delays computed by PPR indicate
that 1 of 8 timing specifications you provided was not met.
To confirm this result, please run xdelay.

# of unrouted connections: 0.

ppr: Generating .LCA File...

*** PPR: WARNING 10604:
An lca file already exists. The old lca file will be saved as
‘calc.lcb’.

ppr: Making Report File...

- ppr @ 1994/01/16 08:55:22 [00:15:57]
= ---- @ 1994/01/16 00:17:33 [00:15:49]

+ ppr required [3896.727] kbytes of dynamic/allocated memory
```

Figure 15-8 Partial PPR Log File for an XC3020APC68-7 Design

Warnings in the PPR Log File

There are several warnings in the PPR log file. Warning 6811 refers to the oscillator loop in the XC3000A design. Since this loop is deliberate, this warning can be ignored.

Warning 7015 reports that the timing specification TS05 does not meet the deadline. That particular constraint was purposely specified so that it would fail in order to illustrate how PPR displays a failure to meet an XACT-Performance specification in the log file. The log file tells you which XACT-Performance specification failed and also gives the timing that it was able to achieve for the associated set of paths. Since PPR checks timing both during placement and routing, this error appears twice. Ignore the timing reported in the warning messages, since these are “best estimates” computed before the placement and routing is complete. The data in the timing analysis summary near the end of the log file is more accurate. The rest of the XACT-Performance constraints all meet the timing requirements.

Warning 10604 occurs whenever PPR saves a new LCA file and there is already an existing LCA file. Since PPR routes the design more than once, these warnings occur in every ppr.log file and can safely be ignored.

Timing Analysis Summary

The tabular timing analysis summary near the end of the ppr.log file shows all XACT-Performance specifications in the design. For each specification, it reports both the timing requirement and the resulting timing for the worst-case path. Any missed specification, such as TS05 in this example, is flagged.

Using XDelay, the Timing Analysis Program

The next step is to verify the timing of your routed design using XDelay.

XDelay is a static timing analysis tool that reports the worst-case timing delays of a routed FPGA design. XDelay has three operating modes:

- Analyze mode quickly shows the maximum clock speed of a design. It reports the worst-case timing paths for each of four typical design path types: pad-to-setup, clock-to-setup, clock-to-pad, and pad-to-pad.
- XDelay-TimeSpec mode verifies which XACT-Performance constraints are met and reports all missed paths in detail.
- XDelay mode provides detailed path timing information according to the selected options and offers insight as to which paths in the design are the most critical. This information helps you determine where to make modifications to meet the design timing requirements.

XDelay has many command options. The following options are the most commonly used. Many of them are demonstrated in this tutorial.

The following options apply to all three operating modes:

- The Flagblk option flags certain blocks to which the path-delay calculator gives special consideration.

- Query / Clear / Save / Read Template are commands to view, clear, save, and load a template file containing XDelay options.

The FailedSpec and SelectSpec options apply to the XDelay-TimeSpec mode:

- The FailedSpec option reports path delays that did not meet an XACT-Performance timing specification. It considers only the specifications selected using the SelectSpec option.
- The SelectSpec option allows you to select from a list of all defined XACT-Performance specifications. The number of delay paths reported for each selected specification is controlled by the TSMaxpaths option.

The following options apply to the XDelay and Analyze modes:

- The ClockToSetup option constrains reporting to paths that start at clocked outputs, such as flip-flop outputs, and end at clocked inputs, such as flip-flop data inputs. Reported delays include the setup requirement on flip-flops.
- The ClockToPad option constrains reporting to paths that start at clocked outputs, such as flip-flop outputs, and end at output pads.
- The PadToSetup option constrains reporting to paths that start at input pads and end at clocked inputs, such as flip-flop data inputs. Reported delays include the setup requirement on flip-flops.
- The PadToPad option constrains reporting to all paths that start at input pads and end at output pads, with only combinatorial logic elements in the path. (This option is not demonstrated, because the only unclocked paths in the tutorial design are in the clock oscillator in the XC3000A.)
- The FromFF and ToFF options together allow reporting on specific paths by selecting the flip-flops at the endpoints of the paths.

For more information on using XDelay, refer to the “XDelay” section of the *XACT Reference Guide.*

Analyzing the Calc Design

This section analyzes the results of the XACT-Performance design created earlier in this tutorial. The routed LCA file input to XDelay contains actual timing delays as well as XACT-Performance specifications. XDelay analyzes this information and shows different types of delay paths according to the options you select. Selected options are stored as an XDelay template file with an .xtm extension. This section demonstrates a typical XDelay analysis command sequence.

Note: The sample XDelay output in this tutorial is from a single Calc LCA file, targeted to the XC3020APC68-7. Your results will vary.

Invoking XDelay

You can invoke XDelay from the operating system prompt, from the Verify menu in XDM, or from the EditLCA program under the XACT Design Editor (XDE), which is not available in the Base Development System.

1. Invoke XDelay from the operating system prompt by typing **xdelay**↵. The XDelay graphic environment appears.
2. Load the Calc design into memory by selecting **Design** → **Design** → **CALC.LCA** from the XDelay menu bar.

Using the Flagblk Option

In the Calc design, there are a number of flip-flops with asynchronous reset signals. You want to ignore the paths through these asynchronous inputs during timing analysis. (Normally a designer is unconcerned with asynchronous reset paths when considering clock-to-setup requirements.) The Flagblk option is useful for specifying that this type of path be ignored.

PPR does not trace paths through the Set and Reset (SD/RD) pins of flip-flops, so the timing results in the timing analysis summary of the ppr.log file do not include any paths through the SD/RD pins.

To verify the timing results from the ppr.log file, you must disable tracing paths through the SD/RD pins. As a result, XDelay does not trace through the above-mentioned paths when calculating path delays, which permits a valid comparison of the timing results from the PPR log file and the XDelay output.

Disabling Paths Through SD/RD Pins of Flip-Flops

Disable these paths with the Flagblk option as follows.

1. Select **`Timing`** → **`Flagblk`** → **`CLB_Disable_SR_Q.`** A menu appears, displaying a list of all CLB blocks in the design.
2. Type **`*`**↵ to select all blocks. A prompt appears, asking whether you are sure you want to select all of the blocks.
3. Select **`Yes`** → **`Done`**. This command disallows paths from a CLB Asynchronous Set or Reset input to the Q output of the flip-flop.

Displaying Current Options

The Query Template command displays the current settings of all options. You can save a template to a file using the Save Template command, load a customized template using the Read Template command, or clear the current template with the Clear Template command.

1. Select **`Timing`** → **`QueryTemplate`** to view the current XDelay options.
2. After viewing the template listing, press any key to return to the XDelay executive screen.

The current template appears on the screen. The template includes all restrictions you applied to each CLB. A partial template for an example XC3000A design is shown in the following figure.

```
XDelay -NoSourceClock
XDelay -NoDestClock
XDelay -NoIgnorenet
XDelay -NoNetfilter
XDelay -NoBreakLoop

Flagblk CLB_Disable_SR_Q ALU/DATA0
Flagblk CLB_Disable_SR_Q STACK2
.
Flagblk CLB_Disable_SR_Q ALU/ENOV
Flagblk CLB_Disable_SR_Q CTL2
```

Figure 15-9 Partial Template for an XC3020APC86-7 Design

Using Analyze Mode

Now that you have restricted XDelay to considering only clocked paths, perform a quick analysis to determine the worst-case timing for the Calc design.

By default, XDelay output is written to the screen. Normally, you want to keep a copy of the analysis results. Also, writing to a file is much faster than reporting to the screen, since there is no need to funnel information through a graphical interface.

1. Select **`Misc → Report`** to specify that you want the results written to a file. You are prompted to enter a file name.
2. Type ↵ to accept the default file name, calc.xrp.
3. Select **`Analyze → Done`**.

For the example 3020APC68 design, XDelay reports the same combinatorial logic loop detected by PPR. Since this loop is deliberately included to create an oscillator, ignore this message.

Examining Analyze Mode Output

Examine the Analyze output file.

1. Press any key to return to the XDelay graphic screen.
2. Use any text editor to examine the XDelay report file, calc.xrp, which was created in the present design directory.

A partial report file for the XC3020APC68-7 example design is shown in the following figure. The last line of the file shows that the design will operate at approximately 12.7 MHz under worst-case conditions.

The XRP file lists the worst-case pad-to-pad, pad-to-setup, clock-to-pad, and clock-to-setup delays. It also provides an estimate of the minimum clock period and maximum clock speed for the input design.

```
Warning: Combinational logic loop(s) have been detected.
These may cause subtle design problems, and may yield some
inaccurate path delays. For a detailed enumeration of these
loops, use the “DRC -Inform” command from within XDE/EditLCA.

XDelay Report File:

Worst case Pad to Pad path delay   : 37.8ns (1 block level)
 Pad “OSC_3K/CQ” (P14) to Pad “OSC_3K/CQL” (P12.T)
```

```
Clock net “CLK” path delays:

Pad to Setup                       : 14.0ns (0 block levels)
 (Includes an external input margin of 0.0ns.)
 Pad to Input FF Setup, Pad “SW7/SW0_P” (P24).
 Target InFF drives output net “SW0”

Clock to Pad                       : 48.3ns (2 block levels)
 (Includes an external output margin of 0.0ns.)
 Clock to Q, net “ADDR0” to Pad “LED/LED1_P” (P30.O)

Clock to Setup (same edge)         : 78.6ns (6 block levels)
 Clock to Q, net “ADDR0” to FF Setup (D) at “ALU2.A”
 Target FFX drives output net “ALU2”

Minimum Clock Period               : 78.6ns

Estimated Maximum Clock Speed      : 12.7Mhz
```

Figure 15-10 Analyze Output for an XC3020APC68-7 Design

Using XDelay-TimeSpec Mode

Use the XDelay-TimeSpec mode to evaluate the timing of your design with respect to the XACT-Performance attributes that you added to your schematic.

The FailedSpec and SelectSpec options are particularly useful for evaluating XACT-Performance results. FailedSpec reports all path delays that do not meet timing specifications, and SelectSpec allows you to select which XACT-Performance specifications you wish to be considered.

Note: Used without the FailedSpec option, the SelectSpec option reports the worst paths for each XACT-Performance specification.

As in the “Using the Analyze Mode” section, create a written report file.

1. Select **`Misc → Report`**.
2. Type **`calcts.xrp`**↵ to distinguish the new output file from the previous one.
3. Select **`XDelay-TimeSpec`**. A menu appears displaying available delay options. The default options are highlighted.
4. Select **`ClearOptions`** to remove any delay options you may have previously selected. The Flagblk options you set earlier in the

tutorial are not cleared, because they are template options rather than delay options.

5. Select the **-FailedSpec** option.
6. The **SelectSpec** option is already on by default, but select it anyway. A menu of defined XACT-Performance attributes appears. You can select any or all of the timing specifications on this list. By default, all XACT-Performance specifications are already selected.
7. Select **Cancel** to accept the list with all entries selected. Next, set the maximum number of paths shown for each failed XACT-Performance constraint to be three.
8. Select **-TSMaxpaths**.
9. Type **3**↵.

Note: If you do not set the TSMaxpaths option, the report file lists delays for every path controlled by each XACT-Performance specification in your design. This may cause XDelay to run out of memory; if not, it produces a very large output file.

10. Select **Done** to initiate the timing analysis. As before, for the example 3020APC68 design XDelay reports the same combinatorial logic loop detected by PPR. Since this loop is deliberately included to create an oscillator, ignore this message.

Examining XDelay-TimeSpec Mode Output

The XDelay-TimeSpec output file contains a great deal of information.

1. Press any key to return to the XDelay graphic screen.
2. Use any text editor to examine the XDelay report file, calcts.xrp.

A portion of the resulting report file for the XC3020APC68-7 example design is shown in the following figure.

```
XDelay: calc.lca (3020APC68-7), xdelay 5.0.0, Mon Jan 17
11:15:36 1994

Warning: Combinational logic loop(s) have been detected.
These may cause subtle design problems, and may yield some
inaccurate path delays. For a detailed enumeration of these
loops, use the “DRC -Inform” command from within XDE/EditLCA.
```

```
XDelay Report File:

Xdelay path report options:

TimeSpec ‘TSO3’ from group ‘FFGRP’ to group ‘PADS’ is
1000.0ns.
TimeSpec ‘TS01’ from group ‘FFS’ to group ‘FFS’ is 2000.0ns.
TimeSpec ‘TS02’ from group ‘PADS’ to group ‘FFS’ is 1000.0ns.
TimeSpec ‘TS04’ from group ‘INFFS’ to group ‘FFS’ is 80.0ns.
TimeSpec ‘TS05’ from group ‘CTL_ADR_FF’ to group ‘ALUFF’ is
50.0ns.
TimeSpec ‘TS06’ from group ‘CTL_ALU_FF’ to group ‘STACKER’ is
30.0ns.
TimeSpec ‘TS07’ from group ‘STACKER’ to group ‘LEDPADS’ is
50.0ns.
TimeSpec ‘TS08’ from group ‘ALUFF’ to group ‘PADS’ is 45.0ns.

TimeGroup ‘ALUFF’ contains these Flip-Flop output nets:
ALU0 ALU1 ALU2 ALU3 OFL

TimeGroup ‘CTL_ADR_FF’ contains these Flip-Flop output nets:
ADDR0 ADDR1

TimeGroup ‘CTL_ALU_FF’ contains these Flip-Flop output nets:
ADDR0 ALU0 ALU2 CONTROL/STATMACH/OTHER OFL
ADDR1 ALU1 ALU3 CONTROL/STATMACH/PUSH WE
.
.
.
TimeGroup ‘STACKER’ contains these Flip-Flop output nets:
STACK/A0 STACK/A3 STACK/B2 STACK/C1 STACK/D0 STACK/D3
STACK/A1 STACK/B0 STACK/B3 STACK/C2 STACK/D1
STACK/A2_1 STACK/B1 STACK/C0 STACK/C3 STACK/D2

Only paths that do not meet the selected TimeSpecs will be
reported.
Output will be sorted by decreasing path delays.
A maximum of 3 paths will be reported for each TimeSpec.

------------------------------------------------------------

TimeSpec ‘TSO3’ summary:
From TimeGroup ‘FFGRP’
To TimeGroup ‘PADS’

TimeSpec limit is          : 1000.0ns (Spec speed = 1.0MHz)
Worst path delay is        : 48.3ns   (Real speed = 20.7MHz)

TimeSpec passes by : 951.7ns

List of delay paths that fail the TimeSpec:
There are no paths that fail the TimeSpec.

------------------------------------------------------------
```

```
TimeSpec ‘TS01’ summary:
From TimeGroup ‘FFS’
To TimeGroup ‘FFS’

TimeSpec limit is          : 2000.0ns (Spec speed = 0.5MHz)
Worst path delay is        : 78.6ns   (Real speed = 12.7MHz)

TimeSpec passes by : 1921.4ns

List of delay paths that fail the TimeSpec:
There are no paths that fail the TimeSpec.

------------------------------------------------------------

TimeSpec ‘TS02’ summary:
From TimeGroup ‘PADS’
To TimeGroup ‘FFS’

TimeSpec limit is          : 1000.0ns (Spec speed = 1.0MHz)
Worst path delay is        : 29.4ns   (Real speed = 34.0MHz)

TimeSpec passes by : 934.6ns

List of delay paths that fail the TimeSpec:
There are no paths that fail the TimeSpec.

------------------------------------------------------------

TimeSpec ‘TS04’ summary:
From TimeGroup ‘INFFS’
To TimeGroup ‘FFS’

TimeSpec limit is          : 80.0ns   (Spec speed = 12.5MHz)
Worst path delay is        : 65.4ns   (Real speed = 15.3MHz)

TimeSpec passes by : 14.6ns

List of delay paths that fail the TimeSpec:
There are no paths that fail the TimeSpec.

------------------------------------------------------------

TimeSpec ‘TS05’ summary: *** TimeSpec FAILED! ***
From TimeGroup ‘CTL_ADR_FF’
To TimeGroup ‘ALUFF’

TimeSpec limit is          : 50.0ns   (Spec speed = 20.0MHz)
Worst path delay is        : 78.6ns   (Real speed = 12.7MHz)

*** TimeSpec FAILS by : 28.6ns ***

List of delay paths that fail the TimeSpec:

Logical Path                              Delay  Cumulative
------------                              -----  ----------

Source clock net : “CLK” (Rising edge)
```

```
From: Blk ADDR1          CLOCK to FE.Y    : 4.5ns ( 4.5ns)
Thru: Net ADDR0               to HH.C    : 8.9ns ( 13.4ns)
Thru: Blk STACK/$1I15/M01     to HH.X    : 5.1ns ( 18.5ns)
Thru: Net STACK/$1I15/M01     to HF.C    : 3.1ns ( 21.6ns)
Thru: Blk STACK0              to HF.X    : 5.6ns ( 27.2ns)
Thru: Net STACK0              to HB.A    : 4.7ns ( 31.9ns)
Thru: Blk ALU/DATA0           to HB.X    : 5.1ns ( 37.0ns)
Thru: Net ALU/DATA0           to ED.D    : 6.2ns ( 43.2ns)
Thru: Blk ALU/$1I308/C0       to ED.X    : 5.1ns ( 48.3ns)
Thru: Net ALU/$1I308/C0       to BE.E    : 4.5ns ( 52.8ns)
Thru: Blk ALU/$1I308/C1       to BE.X    : 5.1ns ( 57.9ns)
Thru: Net ALU/$1I308/C1       to CD.A    : 2.5ns ( 60.4ns)
Thru: Blk ALU/$1I308/C2       to CD.X    : 5.1ns ( 65.5ns)
Thru: Net ALU/$1I308/C2       to DD.B    : 1.9ns ( 67.4ns)
Thru: Blk LU/MUXBLK5/$1I5/M01  to DD.X    : 5.6ns ( 73.0ns)
Thru: Net LU/MUXBLK5/$1I5/M01  to DE.B    : 0.6ns ( 73.6ns)
To: FF Setup (D), Blk ALU3                : 5.0ns ( 78.6ns)

Target FFX drives output net “ALU3”
Dest clock net : “CLK” (Rising edge)
Clock delay to Source clock pin : 2.8 ns
Clock delay to Dest clock pin : 2.8 ns
Clock net “CLK”, delta clock delay [skew] : 0.0 ns

.
. (TWO MORE FAILED PATHS OMITTED)
.
------------------------------------------------------------

TimeSpec ‘TS06’ summary:
From TimeGroup ‘CTL_ALU_FF’
To TimeGroup ‘STACKER’

TimeSpec limit is          : 30.0ns   (Spec speed = 33.3MHz)
Worst path delay is        : 24.0ns   (Real speed = 41.8MHz)

TimeSpec passes by : 6.0ns

List of delay paths that fail the TimeSpec:
There are no paths that fail the TimeSpec.

------------------------------------------------------------

TimeSpec ‘TS07’ summary:
From TimeGroup ‘STACKER’
To TimeGroup ‘LEDPADS’

TimeSpec limit is          : 50.0ns   (Spec speed = 20.0MHz)
Worst path delay is        : 43.4ns   (Real speed = 23.0MHz)

TimeSpec passes by : 6.6ns

List of delay paths that fail the TimeSpec:
```

```
There are no paths that fail the TimeSpec.

------------------------------------------------------------

TimeSpec ‘TS08’ summary:
From TimeGroup ‘ALUFF’
To TimeGroup ‘PADS’

TimeSpec limit is          : 45.0ns   (Spec speed = 22.2MHz)
Worst path delay is        : 40.0ns   (Real speed = 25.0MHz)

TimeSpec passes by : 5.0ns

List of delay paths that fail the TimeSpec:
There are no paths that fail the TimeSpec.

------------------------------------------------------------

Paths not used in TimeSpecs :

There are no paths in this section!

------------------------------------------------------------
```

Figure 15-11 XDelay-TimeSpec Output for XC3020APC68-7

The first section of the XRP file lists all XACT-Performance specifications applied to your design. If you do not specify any paths that fall into a given default path type, FFS:TO:FFS, PADS:TO:FFS, or FFS:TO:PADS, that default specification is set to “auto,” which means that PPR assigns some reasonable value as the timing specification.

The output file then lists the contents of each time group that you defined using TNM attributes and TIMEGRP symbol attributes. This section can be useful in verifying your time group definitions.

The ALUFF time group contains the four ALU outputs and the OFL flip-flop output. Therefore, the group contains all of the flip-flops in the ALU, and no other flip-flops, just as expected.

The CTL_ADR_FF set was defined in the “Grouping Sets with TIMEGRP” section. It is defined as CTLFF:EXCEPT:STFF and should contain only the flip-flops in the CB2CLED macro below CONTROL. The outputs of the CB2CLED macro are ADDR0 and ADDR1, and, as shown in the output, the set CTL_ADR_FF does include only these two flip-flop outputs.

The worst path delay is then reported for each XACT-Performance specification.

For TS05, which missed the target timing, the XRP file includes a detailed listing of the three slowest paths. You can use this listing to examine your critical paths and determine why each path is not routable with the current timing requirement, then take steps to remedy the situation.

For example, for the failed path shown in the output, the longest delay on the path is an 8.9 ns delay between the Y output of CLB FE and the C input of CLB HH. (The block name of CLB FE is ADDR1, since the block is named after the X output, but the signal you are tracing is ADDR0.) Since these CLBs are some distance from each other, the net delay is significant. Compare this net delay to the net delay listed further down the path, between the X output of CD and the B input of CLB DD; the delay between these adjacent CLBs is only 1.9 ns. You might be able to speed up this path by using floorplanning techniques to place the logic in a smaller area.

A comparison of the results of the FailedSpec output with the PPR log file shows that the PPR and XDelay results vary by only a few tenths of nanoseconds. When there is a discrepancy between the two, the XDelay results are correct.

For example, in the FailedSpec output, the last XACT-Performance specification listing is for TS08, FROM:ALUFF:TO:PADS=45. The FailedSpec output shows that the worst path delay is 40.0 ns. The ppr.log file shows the worst path delay to be 40.1 ns.

Using XDelay Mode

In XDelay mode, you can generate reports either based on general path types or analyzing designated paths from specific sources to specific destinations.

Reporting by Path Type

To simplify the analysis of designs without XACT-Performance specifications, the XDelay mode offers four options that restrict reporting to paths of four common types. These options are ClockToSetup, ClockToPad, PadToSetup, and PadToPad. You can select one or more of these options.

The number of paths reported depends on the value of the Maxpaths option.

Use the ClockToSetup option to report the single slowest path between two flip-flops clocked by the same edge of the clock.

1. Select **Misc → Report**. You are prompted to enter the name of the report file.
2. Type **calcc2s.xrp**↵.
3. Select **XDelay**. A menu appears displaying available delay options.
4. Select **ClearOptions** to remove any delay options you may have previously selected.

   The Flagblk options you set earlier in the tutorial are not cleared, because they are template options rather than delay options.
5. Select **-ClockToSetup**.
6. Select **-Maxpaths** and type **1**↵.

   Setting Maxpaths=1 directs XDelay to report only the worst clock-to-setup path in the design.

Note: If you do not set the Maxpaths option, the report file lists delays for every path in your design. This may cause XDelay to run out of memory; if not, it produces a very large output file.

7. Select **Done** and press any key to return to the XDelay graphic screen.
8. Use any text editor to examine the XDelay report file, calcc2s.xrp.

A portion of an example calcc2s.xrp file is shown in the figure below.

The only paths reported are those between flip-flops: in other words, the paths that fall into the ClockToSetup or FROM:FFS:TO:FFS category. Since Maxpaths was set to one, only the worst-case clock-to-setup path is reported.

```
Output will be sorted by decreasing path delays.
Report file may include Clock To Setup paths.
A maximum of 1 path will be reported for each TimeSpec.

------------------------------------------------------------

Logical Path                                Delay Cumulative
------------                                ----- ----------

Source clock net : “CLK” (Rising edge)
```

```
From: Blk ADDR1      CLOCK   to FE.Y      : 4.5ns ( 4.5ns)
Thru: Net ADDR0              to HH.C      : 8.9ns ( 13.4ns)
Thru: Blk STACK/$1I15/M01    to HH.X      : 5.1ns ( 18.5ns)
Thru: Net STACK/$1I15/M01    to HF.C      : 3.1ns ( 21.6ns)
Thru: Blk STACK0             to HF.X      : 5.6ns ( 27.2ns)
Thru: Net STACK0             to HB.A      : 4.7ns ( 31.9ns)
Thru: Blk ALU/DATA0          to HB.X      : 5.1ns ( 37.0ns)
Thru: Net ALU/DATA0          to ED.D      : 6.2ns ( 43.2ns)
Thru: Blk ALU/$1I308/C0      to ED.X      : 5.1ns ( 48.3ns)
Thru: Net ALU/$1I308/C0      to BE.E      : 4.5ns ( 52.8ns)
Thru: Blk ALU/$1I308/C1      to BE.X      : 5.1ns ( 57.9ns)
Thru: Net ALU/$1I308/C1      to CD.A      : 2.5ns ( 60.4ns)
Thru: Blk ALU/$1I308/C2      to CD.X      : 5.1ns ( 65.5ns)
Thru: Net ALU/$1I308/C2      to DD.B      : 1.9ns ( 67.4ns)
Thru: Blk LU/MUXBLK5/$1I5/M01 to DD.X     : 5.6ns ( 73.0ns)
Thru: Net LU/MUXBLK5/$1I5/M01 to DE.B     : 0.6ns ( 73.6ns)
To: FF Setup (D), Blk ALU3                : 5.0ns ( 78.6ns)
Target FFX drives output net “ALU3”

Dest clock net : “CLK” (Rising edge)
Clock delay to Source clock pin : 2.8 ns
Clock delay to Dest clock pin : 2.8 ns
Clock net “CLK”, delta clock delay [skew] : 0.0 ns

------------------------------------------------------------
```

Figure 15-12 ClockToSetup Output for XC3020APC68-7 Design

Specifying Source and Destination

For a report of a specific path, use the FromFF and ToFF options in the XDelay mode. For instance, suppose you are concerned about the path delay between Delay1 and Delay2 in the debounce circuit of the Calc design. (These nets are the outputs of consecutive flip-flops. XDelay can not report path delays that span more than one flip-flop.)

Follow the steps below to get a detailed report of the delays on this path.

1. Select **`Misc → Report`**.

   You are prompted to enter the name of the report file.

2. Type **`dpath.xrp`**↵.

3. Select **`XDelay → -ClearOptions`**.

   You are ready to enter the endpoints of the desired path.

4. Select **-FromFF**.

   A list of all flip-flops in the design appears on the screen. The flip-flops are identified by the name of the output net.

5. Select the source flip-flop by the name of the output net, **DEBOUNCE/DELAY1**.

6. Select **Done** to accept the list of source flip-flops.

   Alternatively, you could select additional flip-flops to get reports on more than one path.

7. Select **-ToFF**.

8. Select the destination flip-flop by the name of the output net, **DEBOUNCE/DELAY2**↵.

9. Select **Done**.

   As with the FromFF option, entering more names would enable you to obtain reports of multiple paths.

10. Select **Done** to initiate the analysis.

Note: In the EditLCA design editor, you can select the FromFF and ToFF flip-flops with the mouse. Alternatively, you can type the output net names or select the flip-flops from the menu.

11. Use any text editor to examine the XDelay report file, dpath.xrp.

The dpath.xrp file details a single path, the path between the Delay1 flip-flop and the Delay2 flip-flop, as shown in the figure below. From the rising clock edge on the Delay1 flip-flop to the setup on the input pin of the Delay2 flip-flop, there is a maximum delay of 11.9 ns under worst-case conditions. There is no detectable clock skew between the two flip-flops.

The From, FromIOB, FromAll, To, ToIOB, and ToAll options are all similar in usage to the FromFF and ToFF options demonstrated in this section. Refer to the “XDelay” section of the *XACT Reference Guide* for more details.

Note: When selecting nets in EditLCA for the From and To options, place the cursor on a pin, such as .X, .YQ, .F1, or .G2, that is connected to the desired net. Make sure the correct net name appears in the status window. Click the left mouse button to add the net to the list of selected nets.

```
From FF “DEBOUNCE/DELAY1”
To FF “DEBOUNCE/DELAY2”
Output will be sorted by decreasing path delays.

------------------------------------------------------------

Logical Path                              Delay  Cumulative
------------                              -----  ----------

Source clock net : “CLK” (Rising edge)
From:  Blk BOUNCE/DELAY1 CLOCK to BA.X    : 4.5ns ( 4.5ns)
Thru:  Net DEBOUNCE/DELAY1     to BA.DI   : 3.4ns ( 7.9ns)
To:    FF Setup (D), Blk DEBOUNCE/DELAY1  : 4.0ns ( 11.9ns)
Target FFY drives output net “DEBOUNCE/DELAY2”

Dest clock net : “CLK” (Rising edge)
Clock delay to Source clock pin : 2.7 ns
Clock delay to Dest clock pin : 2.7 ns
Clock net “CLK”, delta clock delay [skew] : 0.0 ns

------------------------------------------------------------
```

Figure 15-13 Dpath.xrp File for an XC3020APC68-7 Design

Further Reading

Before using XACT-Performance for your own designs, you should read the “XACT-Performance Utility” section of the *XACT Reference Guide*. More information on XDelay can be found in the “XDelay” section of the same manual.

Mentor Graphics Interface/ Tutorial Guide

XEPLD Tutorial

Printed in U.S.A.

Chapter 16

XEPLD Tutorial

Welcome to the Xilinx EPLD Mentor Graphics tutorial. This tutorial will give you a basic understanding of the XEPLD development software and some hands-on experience so you can start your first design as quickly as possible. This tutorial is not meant as a substitute for the *XEPLD Reference Guide* or your Mentor Graphics manuals.

Software Installation

Required Software

The following versions of software are required to perform this tutorial:

- Mentor Graphics Version 8.2_5 or later
- Xilinx / Mentor Graphics Interface DS344 Version 5 or later
- XACT Design Manager (XDM) Version 5 or later

Before Beginning the Tutorial

Before beginning the tutorial, set up your workstation to use Mentor Graphics and XACT Development System software as follows:

1. Verify that your system is properly configured. Consult the release notes that came with your software package for more information.
2. Install the following sets of software:
   - XACT Development System (DS550) version 5.00.
   - Xilinx DS344 Mentor Graphics Version 5 interface.
   - Mentor Graphics software Version 8.2_5 or later, including

Design Manager, Design Architect, QuickSim II, QuickPath, and the ability to produce EDIF netlists from ENWRITE, which requires special licensing.

3. Verify the installation. When you finish the installation, verify that your .cshrc or setup file contains lines similar to the following:

Note: Path names of directories will vary. For more information on paths and environment variables, refer to the release notes that came with your software package.

```
setenv LCA /location_of_ds_344:
setenv XACT /location_of_ds_344:
/location_of_ds550
set PATH=($PATH          \
    $LCA/com/sparc       \
    $LCA/bin/sparc       \
  /location_of_ds550 /bin/sparc\
    )
```

Modifying Mentor Graphics Variables

Make sure that the following Mentor specific variables are set correctly:

- MGC_HOME

  This should point to the Mentor Graphics software tree.

- MGC_GENLIB

  This should point to the Mentor Graphics genlib library, normally $MGC_HOME/gen_lib.

- LD_LIBRARY_PATH

  This variable is used by the Mentor Design DataPort (DDP) routines that are accessed by some Xilinx programs. This variable is normally set with the following, on a SPARC station, assuming that OpenWindows is installed in /usr:

  setenv LD_LIBRARY_PATH $MGC_HOME/shared/lib:$MGC_HOME/lib:/usr/openwin/lib

- MGC_LOCATION_MAP

  This variable should point to a valid location map file.

Every symbol and schematic in your design contains a reference. A reference indicates where the design object resides on your disk. The tutorial designs use variables in their reference definitions so they can be easily relocated. All of the tutorial designs use the variable $xilinx_tutorial to define the path reference. $xilinx_tutorial must be defined in the file pointed to by $MGC_LOCATION_MAP. For example, the design object rcvrsub in the *install_path*/tutorial/mentor/uarttop directory uses the path reference $xilinx_tutorial/uarttop/rcvrsub to define where it is located in the directory structure. If the tutorial directories were copied to the path /home/bclinton/mentor the following two lines must be added to the file pointed to by $MGC_LOCATION_MAP:

```
$xilinx_tutorial
/home/bclinton/mentor
```

If a query was made to determine where the design object '$xilinx_tutorial/uarttop/rcvrsub' is located, the Mentor Graphics tools would use this definition to determine that rcvrsub is at /home/bclinton/mentor/uarttop/rcvrsub.

It is also important that the $LCA variable be instantiated, but not defined, in the file pointed to by $MGC_LOCATION_MAP. To do this, add the following line to MGC_LOCATION_MAP, followed by an empty line:

```
LCA
(empty line)
```

- MGC_WD

  This variable points to the working directory. For the tutorial, it should be set to point to the directory where the tutorial will be worked on.

How to Follow this Tutorial

The tutorial consists of six sessions. Each session begins with an overview and summary of the interactive steps required.

If you are not familiar with Mentor Graphics software, it will take about six hours to complete the entire tutorial. If you are familiar with Mentor Graphics software and you wish to skip the schematic capture and simulation sessions, you can finish in about an hour. We supply example files, including a schematic of the tutorial design.

When you see an arrow between two commands, “→”, it indicates that you select the first command and then select the second command from a menu that the first command displays. For example, **`File → Save Sheet`** means that you select **`File`**, then select **`Save Sheet`** from the menu that appears.

To quit at any time, move the mouse cursor to the square in the upper left corner of the Mentor Graphics window (which is inside the operating system window), press and hold the left mouse button, and select **`Close`** from the displayed menu.

The Edit menu has an Undo command, which allows you to undo changes and deletions.

The Tutorial Design

In this tutorial, you will design the receiver section of a Universal Asynchronous Receiver Transmitter (UART). This circuit converts a serial data stream to parallel bytes and provides handshaking and error detection signals to the host system. The following figure illustrates the functionality of this design.

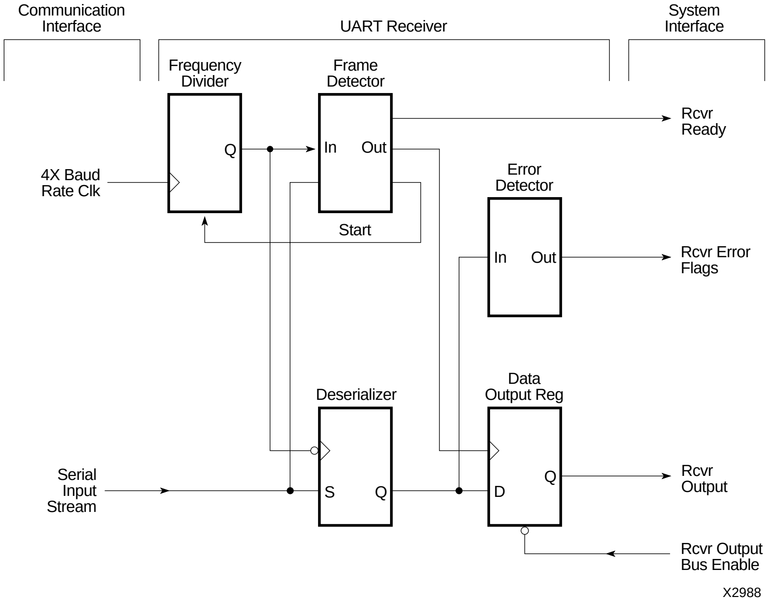


Figure 16-1   UART Receiver Functional Logic Diagram

The example design functions as follows:

1. A serial to parallel shift register (Deserializer) converts the serial stream to parallel data, which is then latched in a register (Output Reg).
2. A simple state machine (Frame Detector) controls the receiver. Once the start bit is detected, the counter (Frequency Divider) begins to count sequentially, clocked by the 4X Baud Rate Clk.
3. The host is notified with the ready signal (Rcvr Ready) and reads the data by asserting the Rcvr Output Bus Enable signal.

4. The output of the counter is decoded to generate the control signals for the shift register, data latch, and error detection circuits. The following signals are generated:

   - Parity Error is generated if a byte parity is odd.
   - Framing Error is generated if any of the stop bits are low.
   - Overrun Error is generated if new data is ready to be latched into the output register before the CPU reads the previous data.

See the following figure for the format of the serial input stream.

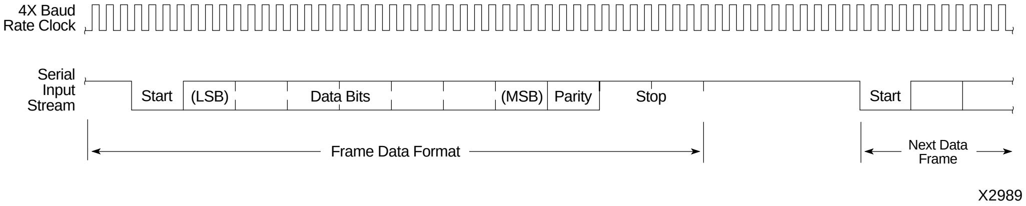


Figure 16-2 UART Serial Input Waveform

The Example Files

All the files containing the UART examples are installed into a sample design directory named $LCA/tutorial/mentor/uart. The design in the uart directory contains a PAL symbol and its associated equation file. A second sample design directory, $LCA/tutorial/mentor/uarttop, contains an alternative implementation based purely on schematic library symbols. The file names in both these directories are listed in the following table. These directories also contain some of the resulting report files.

Table 16-1 Tutorial Directories

Directory *XACT*/tutorial/mentor/uart	
uart	Top level design directory
rcvr.pld	PLD Equations
uart.do	Simulation Commands
Directory *XACT*/tutorial/mentor/uarttop	
uarttop	Top level design directory
rcvrsub	Design directory for rcvrsub
rcvrsub/rcvrsub	Symbol
uarttop.do	Simulation Commands

During the course of this tutorial, you will use these sample files as examples of a completed design. The schematic capture example files are installed during library installation. If these files do not exist, refer to the library installation instructions in the Release Document for directions.

Mentor XEPLD Demonstration Procedure

This section briefly summarizes all the steps needed to run through the entire EPLD tutorial design as a demonstration. These steps are described fully in the following tutorial sessions. This demonstration procedure assumes you have completely installed and configured XEPLD implementation software (DS550), the Mentor Graphics Library and Interface (DS344), and the Mentor Graphics software system.

1. Change to the tutorial directory; the default path is $LCA/tutorial/mentor/uart.
2. Invoke **`pld_dmgr`** at the UNIX prompt.
3. From the Mentor Graphics Design Manager, select the uart container, then select **`Open`** → **`pld_da`**. The Design Architect window appears with the uart schematic window in it.

4. Follow these steps to make the schematic more visible:
   - Click on the square in the upper right corner of the Design Architect window to fully expand it.
   - Expand the schematic window as well.
   - Press Shift-F8 to expand the drawing within the schematic window.
   - Examine the schematic as you wish.
5. To exit Design Architect, move the mouse cursor to the square in the upper left corner of the Design Architect window, press and hold the left mouse button, and select **Close** from the displayed menu.
6. Display the contents of the rcvr.pld file in an operating system window using an ASCII text editor such as vi.
7. From the Design Manager, select the uart container again. Select **Open → pld_men2xnf8**. A dialog box appears. Type **7354-12PC68** in the Part Type field. Select **OK**.
8. Select the uart.xnf icon. Select **Open → pld_xemake**. Follow these steps to run PLD_XEMake:
   - To create an Intel HEX programming file, select the **File** option under Target and enter **uart.prg** in the text box that appears.
   - Select the **Yes** option under Signature and enter **uart** as the signature string in the text box that appears.
   - Select the **OK** button.
9. Select the uart container again, and select **Open → pld_timsim8** to prepare for timing simulation. Select **Auto Generate** under the Schematic option. Select **Yes** under the Run QuickSim option. Select **OK**.
10. When PLD_TIMSIM8 is complete, the QuickSim window appears. Expand the QuickSim window by clicking on the button in the top right corner that contains the larger square.
11. Move the mouse to the large display area, then select **Force → From File** from the pop-up menu. Type **uart.do** in the prompt window.

12. The trace and list files appear. Iconize the trace file by clicking on the button in the top right corner that contains the smaller square.
13. Expand the list file view by clicking on the button in the top right corner that contains the larger square.
14. Use the Page Up and Page Down keys to scroll through the list file. Examine the file as you wish.
15. Iconize the list file and click twice on the trace file icon. Expand the trace file window.
16. Select **`View → All`** from the pop-up menu to make all the waveforms visible.
17. To quit QuickSim, move the mouse cursor to the square in the upper left corner of the Mentor Graphics window, press and hold the left mouse button, and select Close from the displayed menu. A dialog box appears. Select **`Without Saving`** followed by **`OK`**.

Overview of the Sessions

The six sessions cover the following topics:

Session 1: Using the XEPLD Software

Session 2: Drawing the Design in Design Architect

Session 3: Defining the PLD Equations

Session 4: Fitting the Design Using XEPLD

Session 5: Simulating the Design Using QuickSim

Session 6: Completely Schematic-Based Designs and Functional Simulation

Which sessions you should complete depends on whether you need to learn about Mentor Graphics software:

- If you are unfamiliar with Mentor Graphics, you should go through the entire tutorial.
- If you are familiar with Mentor Graphics and want to focus on learning the XEPLD software, go through Sessions 1, 3, and 4 and the last part of Session 6. You might want to glance at Step 1 of Sessions 2 and 5 just to see how XDM interfaces with Mentor Graphics software.

Session 1: Using the XEPLD Software

Session One concentrates on the XEPLD environment. The following operations are introduced in this session:

STEP 1: Prepare the System

STEP 2: Start XDM

STEP 3: Select Menu Items

STEP 4: Configure the XEPLD Environment

Step 1: Prepare the System

You must make sure that you installed all the software, the required libraries, and the tutorial designs and that you have set up the UNIX environment.

1. Install the XEPLD development system, the Mentor Graphics library, and the Xilinx Mentor Graphics Tutorial. Refer to the installation instructions in the Release Notes for information on how to install these products.
2. Make sure the XACT software directory is included in the path and the XACT environment variable in your .cshrc file. (This also allows you to reference the XACT directory with $XACT. When *$XACT* is shown in italics, it means "substitute the directory pointed to by the XACT variable.")
3. If you are using a workstation, copy the tutorial files from the $LCA path established during installation to a user directory, for example:

   ```
   % cd
   % mkdir mentor
   % cd mentor
   % cp -r $XACT/tutorial/mentor/*.
   ```

   You must then edit the location map file (pointed to by the MGC_LOCATION_MAP variable) so that the Mentor Graphics software recognizes the new tutorial location.

   The location map file should have an entry such as the following:

   ```
   $xilinx_tutorial
   /home/bclinton/mentor
   ```

4. Make sure the MGC_WD, MGC_HOME, MGC_GENLIB, and MGC_LOCATION_MAP variables are set according to Mentor Graphics requirements. The setting for MGC_WD should be as follows:

   ```
   cd uart
   setenv MGC_WD /home/bclinton/mentor/uart
   ```

Step 2: Start the Mentor Graphics Design Manager

Start the Design Manager by entering the following:

```
cd /home/bclinton/mentor/uart ↵
```

```
pld_dmgr ↵
```

The Design Manager window is displayed. Its structure is as follows:

- The subwindow on the left is the Tools window, which contains icons for the schematic capture, text editing, simulation, and programmer control environments within the Mentor Graphics software.
- The subwindow in the middle is the Directory window, which contains icons for the design directories, schematic sheets, and other files that are part of your designs.
- The subwindow on the right is the Command Palette. It provides easy access to the most commonly used Design Manager menu items.
- Across the bottom is a listing of the keyboard shortcuts available to you.

To quit at any time, move the mouse cursor to the square in the upper left corner of the Mentor Graphics window (which is inside the operating system window), press and hold the left mouse button, and select **`Close`** from the displayed menu.

Step 3: Select Menu Items

There are two types of menus in Mentor Graphics software: pull-downs and pop-ups.

- The pull-down menus are across the top of the Mentor Graphics window (which is inside the operating system window). To select a command on a pull-down menu, move the mouse cursor to the menu title, press and hold down the left mouse button, slide the mouse until the desired command is highlighted, then release the mouse button.

- To display a pop-up menu, press and hold down the right mouse button. To select a command, slide the mouse until the desired command is selected, then release the mouse button. The pop-up menus are context-sensitive.

The mouse button actions are as follows:

- Select for the left mouse button
- Strokes (commands activated by mouse movements) for the middle mouse button
- Pop-Up Menu for the right mouse button

Session 2: Drawing the Design in Design Architect

In this session you will learn how to draw Mentor Graphics schematics using Design Architect, Mentor Graphics's schematic design entry package. Design Architect contains a large repertoire of features. To be able to effectively use the software, you need only master a few basic operations. These include creating new schematics, editing existing schematics, wiring components together, adding names to signals and components, adding attributes to components, and saving your design.

In this session you will perform the following functions:

STEP 1: Open and View the Existing Design

STEP 2: Create a New Schematic

STEP 3: Change Zoom Level

STEP 4: Enter and Arrange Symbols

STEP 5: Create a Bus

STEP 6: Create Wires

STEP 7: Add Names

STEP 8: Add a Bus Name

STEP 9: Assign Xilinx EPLD Attributes

STEP 10: Finish the Drawing

STEP 11: Assign Signals to Specific EPLD Pins

STEP 12: Check Your Design

STEP 13: Save Your Design

STEP 14: Exit Design Architect

Step 1: Open and View the Existing Design

Select the uart container (folder with schematic symbols on it) in the directory window, press and hold down the right mouse button, and select the **`Open → pld_da`** command.

At first a small process window appears.

After a few minutes, the Mentor Graphics Design Architect window appears. The UART design is displayed in the large central window with the black background.

Click on the square in the upper right corner of the Design Architect window (which is inside the operating system window). The Design Architect window expands to fill the entire screen.

Then expand the schematic window, which is highlighted in light blue and has a black background, by clicking on its square in the upper right corner.

Finally, press Shift-F8 to make the entire design visible.

The UART design schematic appears as in the following figure.

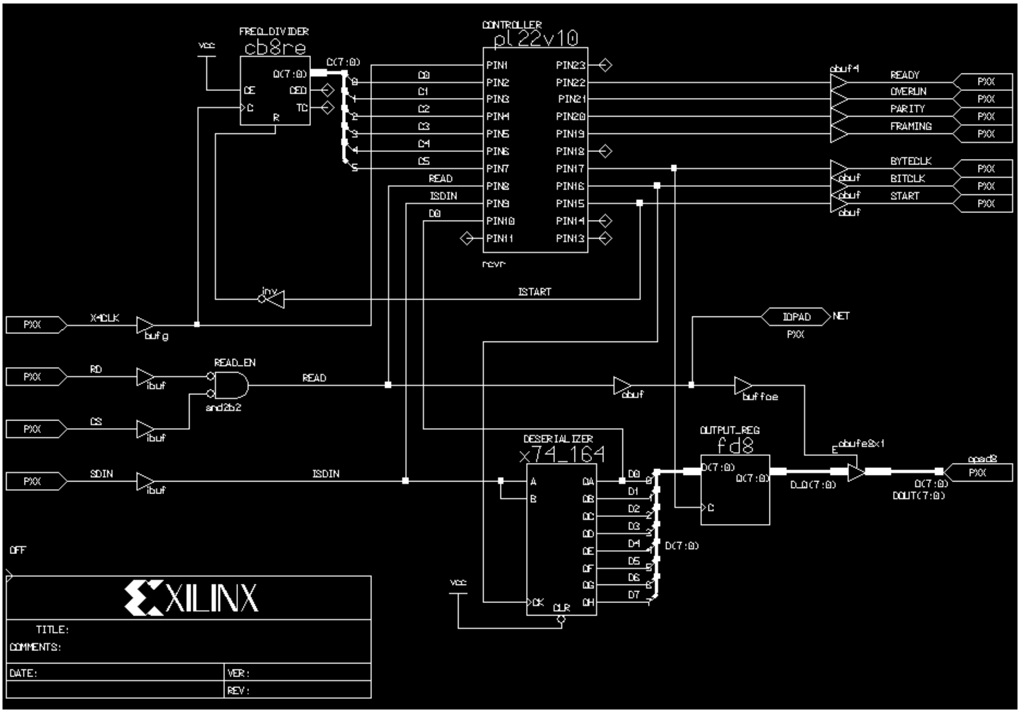


Figure 16-3 The Complete UART Schematic

Three windows appear to the right of the design window:

- The Active Symbol window displays the last symbol added to the schematic, which allows you to add multiple copies of a symbol without using the palette or menus.
- The Palette window displays either icons for commonly used commands or library components that you use to create your design.
- The Context window represents the entire schematic workspace and indicates which part of the entire sheet the main schematic window is displaying.

Take some time and examine the UART schematic. Notice the library PLD, the other standard symbols (counters, shift registers etc.), and the input and output pads.

Note: The library PLD is especially significant because it requires that you use an equation file to define its internal logic. Session 3 explains how to do this.

To close the UART schematic, move the mouse cursor to the square in the upper left corner of the schematic window, press and hold the left mouse button, and select **`Close`** from the displayed menu.

If you are familiar with your schematic capture tool and you do not wish to draw the schematic, close the Design Architect window and skip to Session 3.

Step 2: Create a New Schematic

To open a new schematic window, follow these steps:

1. Select **`File`** → **`Open`** → **`Sheet`**.
2. A dialog box appears. Type **`$xilinx_tutorial/uart/uart2`** to create a design called UART2.
3. Select the **`OK`** button or press Return.
4. Click on the square in the upper right corner of the schematic window to expand it.

Step 3: Change Zoom Level

Design Architect provides several commands for changing the area of the schematic that appears on your screen. These commands allow you to reduce or magnify the area of the schematic displayed:

Zoom In
Zoom Out

Zoom In magnifies the area currently displayed. Zoom Out, its opposite, reduces the scale of the display and displays more of the schematic.

Two commands allow you select a specific area of the schematic displayed:

View Area (hold down F8 key and sweep area with mouse)
View Full (Shift-F8)

View Area lets you select a specific area of the schematic for display. View Full displays the entire schematic in the work space window.

Use the Zoom Out command twice, or until the grid marks in the schematic window disappear.

Step 4: Enter and Arrange Symbols

Before you add symbols, you must display the Xilinx libraries.

To display the library for the XC7000 devices, follow these steps:

1. Select **`Libraries → XACT_LIB`** from the menus.
2. The Palette window changes to a library listing. Select **`UNIFIED_LIB`** from the list.
3. Next, a list of device families appears. Select **`XC7000 LIB`**.
4. Finally, you are given the alternatives **`BY TYPE`** and **`ALL PARTS`**. Select **`ALL PARTS`**. All the XC7000 library symbols are listed.
5. Press and hold the right mouse button while the mouse cursor is in the library listing. The Schematic Palette pop-up window appears. Select **`Show Scroll Bars`** from the menu.

You are now ready to start adding symbols. Add a symbol by following these steps:

1. Scroll the library listing until you can see the ibuf symbol.
2. Click on the **`ibuf`** symbol in the listing.
3. Move the mouse cursor near the center of the left side of the schematic display window. The symbol follows the mouse.
4. Click with the left button. The symbol is now placed.
5. Add two more ibuf symbols below the first one.

In Mentor Graphics software, a symbol remains selected until you deselect it.

Click several times, slowly, on each ibuf symbol. Notice how each becomes unhighlighted, then highlighted. Now press the F2 function key. This key deselects everything in the schematic.

Note: If you double-click on a symbol, you activate the Open Down command. If you do this accidentally, click on the cancel button.

Different pop-up menus appear depending on whether something is selected and what is selected. If you deselect everything with the F2 key, you always see the ADD pop-up menu in the schematic window when you press the right mouse button. The ADD menu is probably the menu you will use most, therefore, you will probably want to use the F2 key often.

After you have added symbols, you can move and delete them. To move a symbol, follow these steps:

1. Select the lower ibuf symbol.
2. Select **Move** from the schematic window's pop-up menu.
3. Move the mouse. Notice how the symbol moves with it.
4. Click the left mouse button. This places the symbol in its new location.
5. Press the F2 key to deselect the symbol.

To delete a symbol, follow these steps:

1. Select the ibuf symbol that you just moved.
2. Select **Delete** from the pop-up menu. The symbol disappears.

If you move or delete something by mistake, you can use the Undo command to reverse the move or deletion.

For example, select **Undo** from the pop-up menu. The ibuf symbol you just deleted reappears.

Select **Undo** again three more times. The symbol is moved to its original position.

Add the following symbols: and2b2, equal, cb8re, inv, four ipads, pl22v10, and vcc.

After all the symbols are added, press the F2 key to deselect everything.

The inv symbol must be rotated. Follow these steps:

1. Select the **inv** symbol. Be sure nothing else is selected.
2. Select **Rotate/Flip** from the pop-up menu twice.
3. Press the F2 key to deselect everything.

When you are finished, the screen should look like the figure below. Move the symbols if necessary, and deselect everything when you are done (press F2).

Note: The bufg symbol represents a FastCLK input. Leave some space between the ipads and the ibufs. When you name a wire between a pad and a buffer, you name the pin.

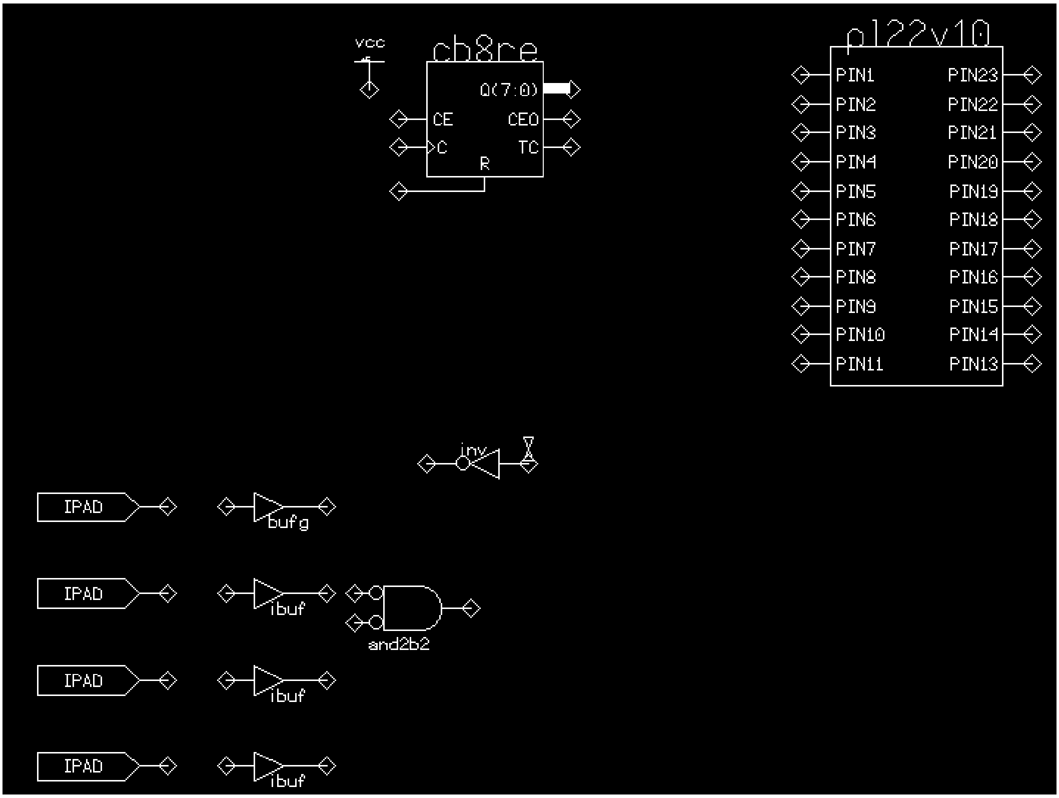


Figure 16-4 Adding Symbols

Step 5: Create a Bus

To create a bus between six of the Q output pins of the cb8re and pins 2 through 7 of the pl22v10, follow these steps:

1. Move the cb8re symbol so that its Q bus is vertically positioned between pins 1 and 2 of the pl22v10.
2. Use the **Bus** command on the pop-up menu to draw a bus in the schematic. The cursor changes to a star. You add bus segments by

clicking once at the beginning of the segment, once every time the segment bends, and twice at the end.

3. Place the cursor on the Q bus.
4. Click the left mouse button.
5. Move the cursor to about a third of the way between the cb8re and the pl22v10 and click once again.
6. Move the mouse cursor down until it is vertically positioned between pins 6 and 7 of the pl22v10.
7. Click the left button twice to end the bus.
8. Select the **`Cancel`** button in the ADD BUS dialog box that extends along the bottom of the screen.
9. Press F2 to deselect everything.

The following figure shows how the schematic appears after you have added a bus. Note the position of the vertical segment in relation to the pins on the pl22v10.

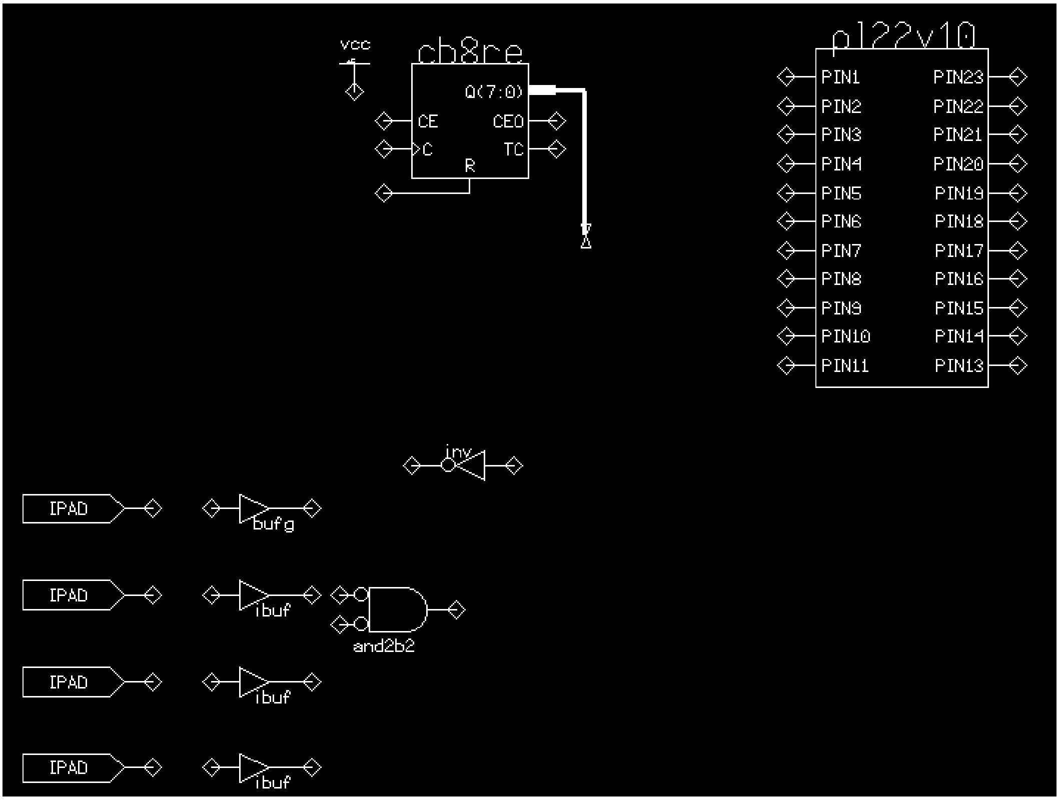


Figure 16-5 Adding a Bus

Step 6: Create Wires

To create a signal between the bus you just created and pin 2 of the pl22v10, follow these steps:

1. Use the **Wire** command on the pop-up menu to draw wires in the schematic. As in the bus command, the cursor changes to a star.
2. Place the cursor on pin 2 of the pl22v10 and click left.
3. Move the cursor straight across to the bus.
4. Double click to complete the wire.

5. Select Cancel from the Add Wire dialog box at the bottom of the screen to stop adding wires.

When you draw wires, Mentor Graphics software automatically connects to the closest pin. If you make a mistake, select and delete the unwanted wire (you delete wires in the same way you delete symbols).

If you connect a wire to a bus, a short diagonal segment named with an “R” appears. This segment, called a “bus ripper”, splits an individual wire from the bus and names the wire. In the section on naming buses, you will learn how to number bus rippers as well.

If the wire segment between the bus and pin 2 of the pl22v10 is not straight, move one of the symbols or bus segments until the wire is straight. If your wires are not straight, multiple pins on the pl22v10 may overlap and appear confusing.

You can quickly add more wires of the same length by copying them. Follow these steps:

1. If it is not already highlighted, select the wire to be copied: in this case, the wire you just created. Select the bus ripper as well.
2. Select the **Copy** command from the pop-up menu.
3. Move the cursor to move the copied wire so it is between the bus and the pin 3 input.
4. Click the left button to place the new wire. Note that yellow circles appear at each end of the wire. This means that the wire is not connected.
5. Repeat steps 2 through 4 to place nets between the bus and pins 4 through 7 of the pl22v10.
6. Press Shift-F6 (Connect All) to connect all the nets.
7. If a segment of the bus extends beyond the last wire connected to it, delete this extra segment (be sure nothing else is selected first).
8. Press F2 to deselect everything.

Note: To move nets, names, or any other objects in your schematic, use the Move command as previously described for moving symbols. To copy any object, use the Copy command.

A junction (a point where three or more wire segments connect) is represented as a square. Design Architect automatically places a junction whenever you terminate one segment onto another.

The following figure shows how the schematic appears after you have added the nets. Add the nets shown.

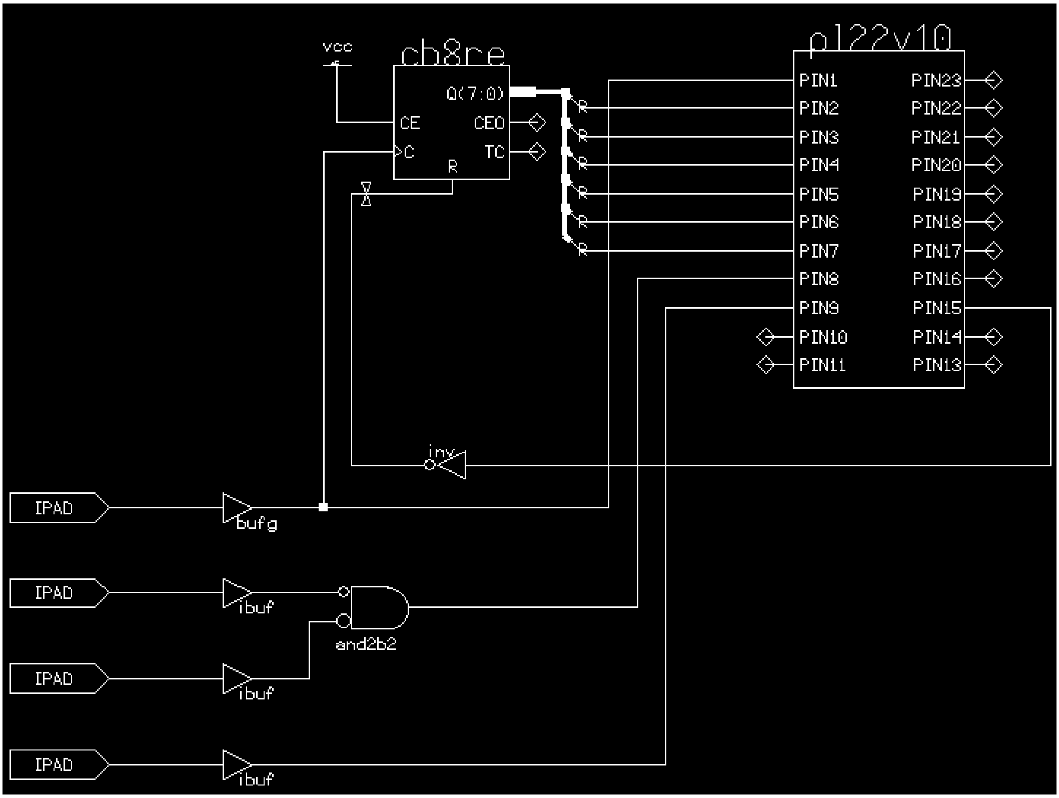


Figure 16-6 Adding Nets

Step 7: Add Names

The reports generated by the XEPLD Fitter refer to all your on-chip signals using names based on your symbol names and symbol output pin names. For example, the macrocell logic and the signal produced by the least-significant bit of the CB8RE counter is referred to in the reports using the name FREQ_DIVIDER:Q0. Assigning your own unique names will make it much easier to identify elements of your design in the reports.

Names assigned to EPLD device pins in the Pinlist report, and names on all signals used during simulation, are taken from the names you place on nets in your schematic. For example, the name X4CLK that you place on the wire between the IPAD and BUFG is listed as a pin name in the Pinlist report produced by the XEPLD Fitter. You will also apply your clock waveform to the X4CLK signal during functional and timing simulation.

Use the **`Properties`** → **`Add`** command to add names to symbols and nets. To name the pl22v10 component, follow these steps:

1. Select the symbol. Be sure nothing else is selected.
2. Select the **`Properties`** → **`Add`** command from the pop-up menu.
3. A dialog box appears. Type **`INST`** in the Property Name box and **`CONTROLLER`** in the Property Value box. Select **`OK`** or press Return.
4. The name appears, attached to the cursor. Position the cursor where you want to place the name, in this case, above the component.
5. Click the left button to place the name.
6. Press F2 to deselect everything.

To name the output wire of the and2b2 symbol, follow these steps:

1. Select the joint where the output of the symbol connects to the wire. Be sure you have selected only that joint — a single star should appear at that location.
2. Select the **`Properties`** → **`Add`** command from the pop-up menu.
3. A dialog box appears. Type **`NET`** in the Property Name box and **`READ`** in the Property Value box. Select **`OK`** or press Return.
4. Position the cursor where you want to place the name, in this case, above the wire.
5. Click the left button to place the name.

If you put a name in the wrong place, you can move it. First place the mouse cursor over the name and press the F1 key. This is the best way of isolating the name. Then move the name just as you would a wire or symbol.

Use only uppercase alphanumeric characters and the underscore “_” in names.

Use the **Properties → Add** command to name the symbols and nets as shown in the following figure:

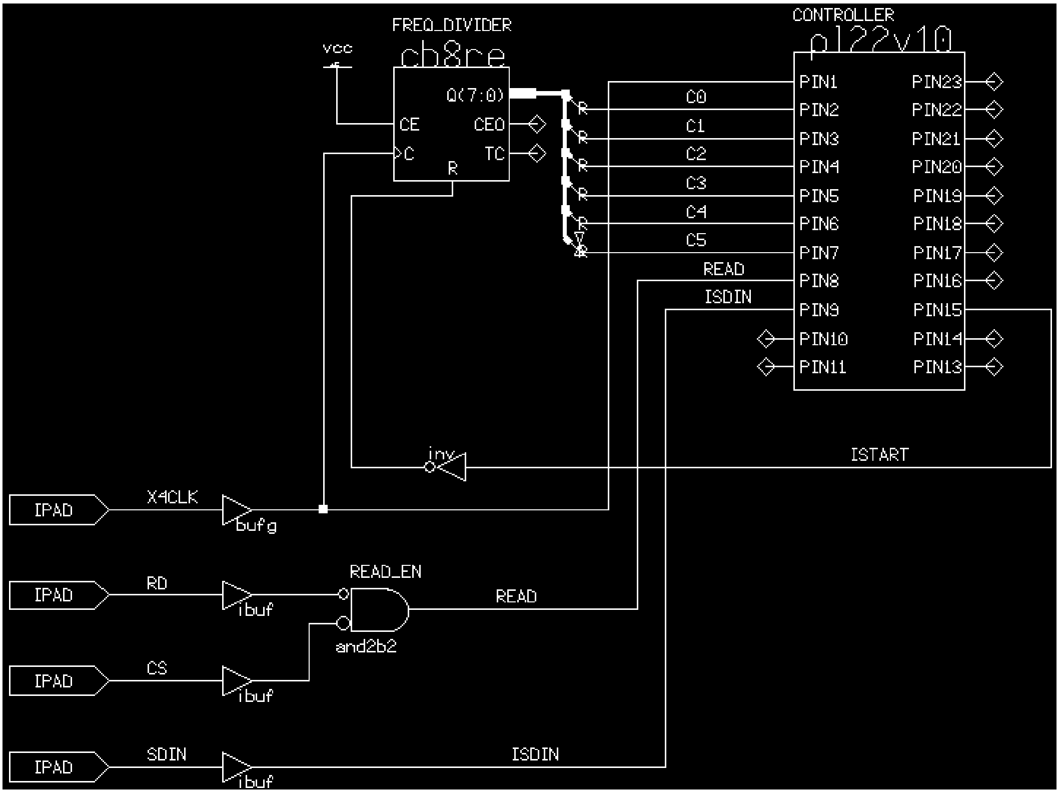


Figure 16-7 Adding Names

The component names are FREQ_DIVIDER and READ_EN. The wire names are as follows: C0, C1, C2, C3, C4, C5, X4CLK, RD, CS, ISTART, SDIN, and ISDIN.

Note: Some of the text in your schematic may be hard to read when the design is finished and you are viewing it all at once. As an option, you can change the text size. Follow these steps:

1. Select the text you want to resize using one of these methods:
   - Use the F1 key to select just the name under the mouse cursor.
   - Press F2 to deselect everything, select the **Select Area → Property** command from the pop-up menu, and sweep an area of the schematic while pressing the left mouse button. All text in the area you swept out is highlighted.
2. Select **Change Height → 1.0** from the pop-up menu. This is the size used in the example UART schematic. You can make the text larger or smaller if you like.

Step 8: Add A Bus Name

A bus must have a name. If a bus is split into individual signals, the numbers you assign to the bus rippers (in place of the "R") must be within the range specified by the bus name. For example, if the bus name is D(7:0), the signal numbers must be 0, 1, ... 7. The order and arrangement of the nets from the bus does not matter.

If you connect a bus to a multi-signal pin, the order in which signals are listed in the bus name determines which signals from the bus are connected to which signals on the pin. For example, if a pin is named Q(7:0) and the bus connected to it is D(7:0), this connects Q0 to D0, Q1 to D1, Q2 to D2, and so on. However, if you name the bus D(0:7), this connects Q0 to D7, Q1 to D6, and so on.

Use the **Properties → Add** command to add a bus name. Follow these steps:

1. Zoom in on the bus: position the mouse cursor just above the cb8re symbol, press and hold down the F8 key, sweep over the bus with the mouse cursor, and release the F8 key.
2. Select the joint that connects the cb8re to the bus. Be sure to select only this joint. A star appears at that joint.
3. Select the **Properties → Add** command from the pop-up menu.
4. In the dialog box that comes up, type **NET** in the Property Name box and **C(7:0)** in the Property Value box. Select OK.
5. Position the cursor where you want to place the name, in this case, above the bus.

6. Click the left button to place the name.
7. Press F2 to deselect everything.

Next, you must number the bus rippers correctly. Follow these steps:

1. Position the cursor on the top right corner of the uppermost “R” and press F1 to select it.

   If you select the wire or ripper segment instead, press F2, move the mouse cursor farther away from what you accidentally selected, and try to select the “R” again.
2. Select the **`Change Values`** command from the pop-up menu.
3. A dialog box appears at the bottom of the screen. Backspace over the “R”, type **`0`** in response to the prompt, then select **`OK`** or press Return.
4. Repeat these steps to name the next bus rippers 1, 2, 3, 4, and 5. The following figure shows how the screen appears after you have changed the bus ripper names.

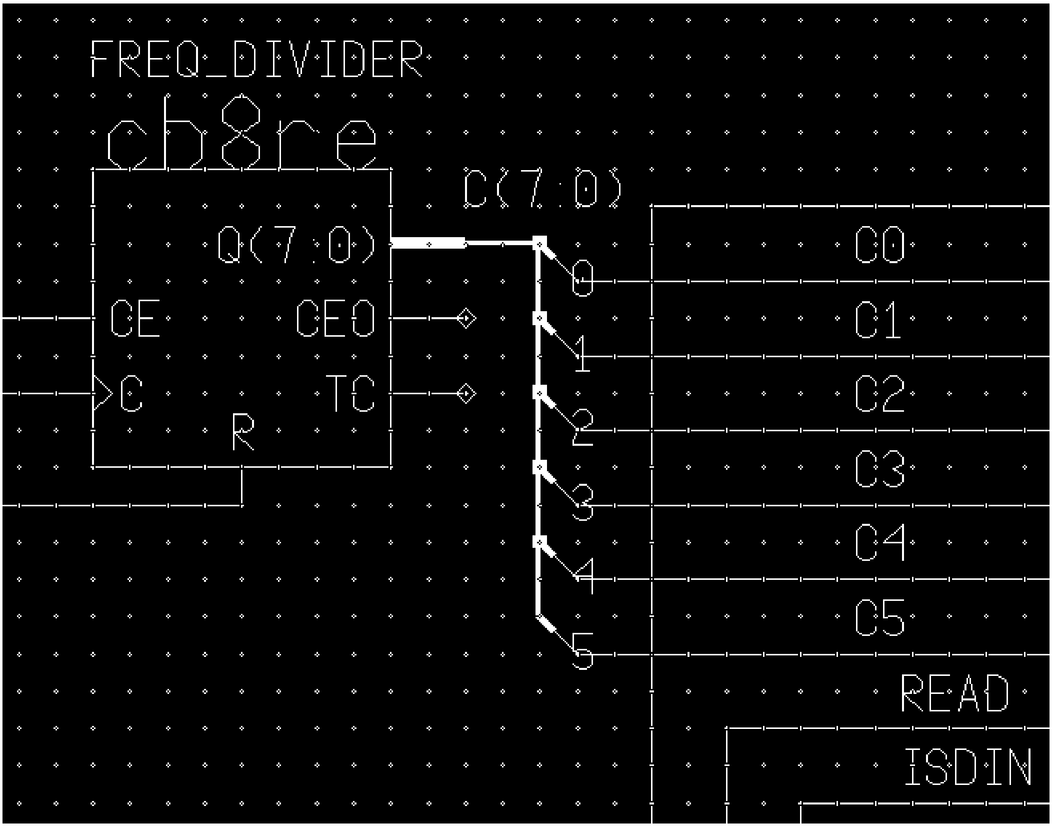


Figure 16-8 Adding a Bus Name and Ripper Names

5. Change the height of the text if you wish.
6. View the entire design again by typing Shift-F8.

Step 9: Assign Xilinx EPLD Attributes

The PLD=*filename* symbol attribute must be attached to each PLD in your schematic. This attribute identifies the bitmap file (prepared by the PLUSASM assembler) used to define the logic of the PLD. The preparation of the logic equations for the PLD is covered in Session 3. Refer to the figure below.

To add the necessary attribute to the pl22v10 component, follow these steps:

1. Select the pl22v10 component.
2. Select the **`Properties`** → **`Add`** command.
3. In the dialog box that comes up, type **`PLD`** in the Property Name box and **`rcvr`** in the Property Value box.
4. Position the cursor where you want to place the name, in this case, below the symbol.
5. Click the left button to place the name.
6. Press F2.

You must also specify a global property. You can attach a global property to any symbol, but typically you attach it to a tblock symbol.

To add the global attributes, follow these steps:

1. Zoom out to make space.
2. Add a tblock symbol in the lower left corner of the design.
3. Make sure only the tblock symbol is selected.
4. Select the **`Properties`** → **`Add`** command.
5. Type **`PRELOAD_OPT`** in the Property Name box and **`OFF`** in the Property Value box. Select **`OK`**.
6. Position the cursor where you want to place the name, in this case, above the symbol.
7. Click the left button to place the name.

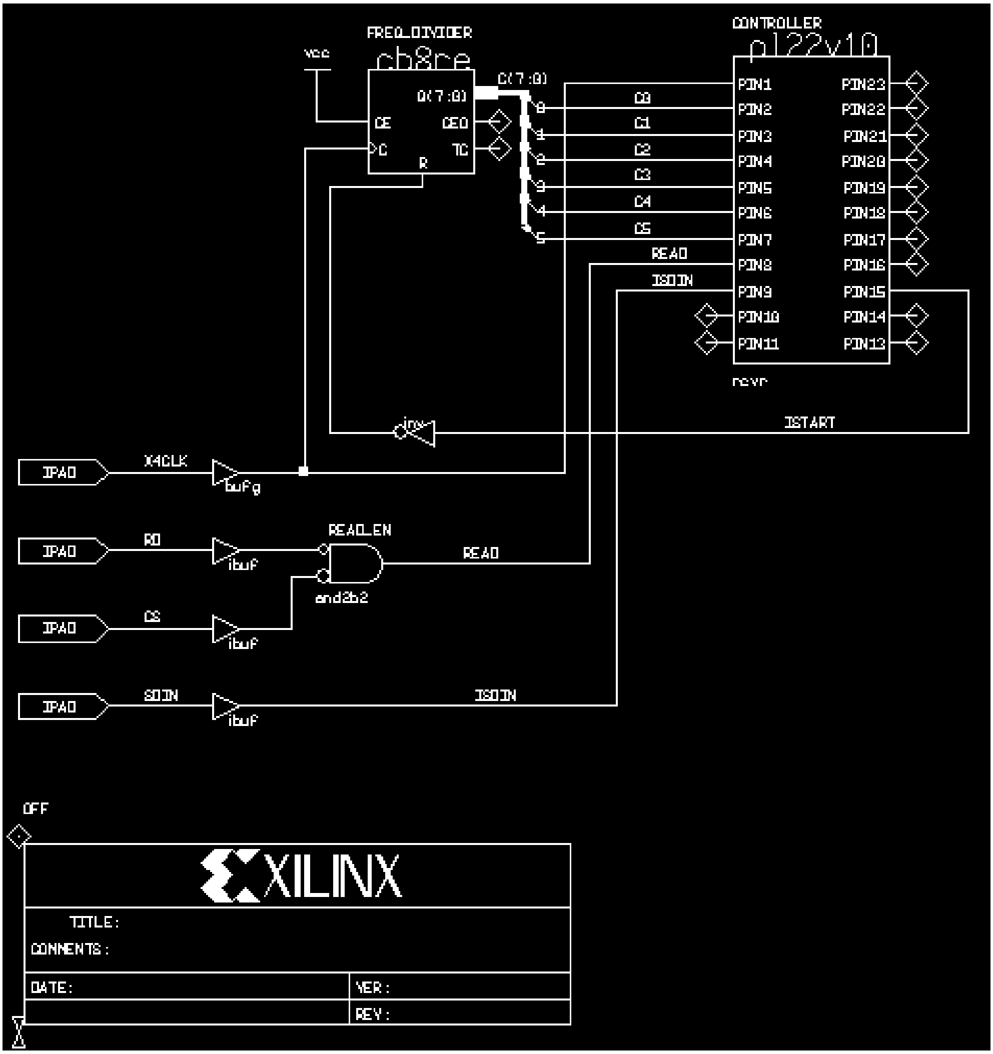


Figure 16-9 Adding Global Attributes

Step 10: Finish the Drawing

You now know enough Design Architect commands to draw schematics on your own. The figure below shows the completed UART receiver schematic.

Use the **`View Full`** (Shift-F8) command to display all of the schematic's logic you have drawn so far. Zoom out and / or scroll right, then add the remaining components, nets, and names as shown in the figure below.

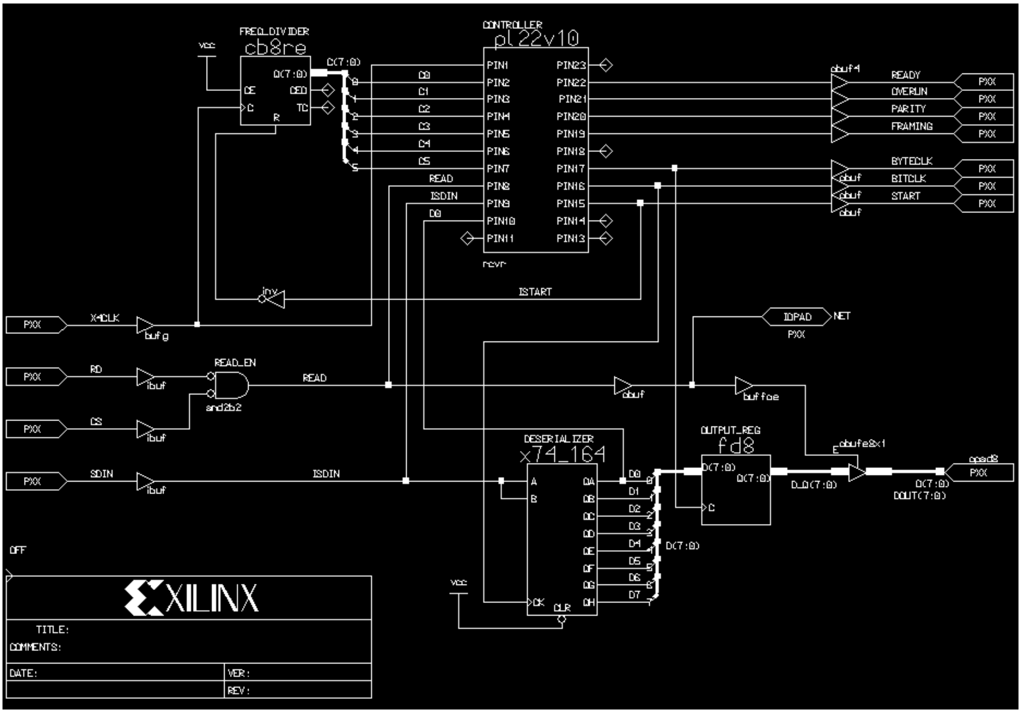


Figure 16-10 The Complete UART Schematic

Note: Be careful to double check names assigned to components and nets in the schematic. Be sure not to confuse characters in names such as "DO" and "D0". Conflicting names on nets cause errors during design compilation.

Step 11: Assign Signals to Specific EPLD Pins

As an option, you can assign signals to specific EPLD pins. To do pin assignment, you must be aware of the architecture of the target EPLD device so that you do not, for example, assign an output to an input pin.

You assign pins to input and output signals by assigning the LOC=*pin_number* attribute to ipad and opad components. For example, you can assign a pin number to the RD and CS signals to ensure that they are next to each other on the chip.

To assign the RD signal to pin 18, follow these steps:

1. Use F1 key to select the Pxx string.
2. Select the **`Change Values`** command from the pop-up menu.
3. In the dialog box that comes up, type **`P18`** in the New Value box. Select **`OK`**.
4. Repeat these steps to assign the CS signal to pin 19.

These attributes assign signals RD and CS to specific pins of the targeted XC7354 EPLD device.

Note: If a pin number is entirely numeric, for example pin 18, you must begin the pin number with a “P”. If the pin number begins with a letter, for example pin E7, the “P” is not needed. You can assign pins to individual IPAD, OPAD, and IOPAD symbols, but not to the multi-bit OPAD8 symbol in this schematic.

The remaining signals are allocated automatically by the XEPLD software.

Step 12: Check Your Design

Use the **`Check → Sheet`** command to verify that your design does not violate any design rules.

After the check, you should see only warnings about unconnected outputs on the cb8re and unconnected pins on the pl22v10. You can ignore these.

Step 13: Save Your Design

Use the **`File → Save Sheet`** command to save your design.

If you wish to use the schematic you created for the rest of this tutorial, you can use the **`File → Save Sheet As`** command and save it to $xilinx_tutorial/uart/uart. Otherwise, use the Xilinx-supplied UART file for the rest of the tutorial.

Note: You must not rename Mentor Graphics schematics using UNIX. The Mentor Graphics security system does not allow you to open or process any renamed schematics. Use the File → Save Schematic command instead.

Step 14: Exit Design Architect

To quit, move the mouse cursor to the square in the upper left corner of the Mentor Graphics window (which is inside the operating system window), press and hold the left mouse button, and select **`Close`** from the displayed menu.

You have now completed this session and are ready to proceed to the next section of the tutorial, where you define the function of the pl22v10 PLD component used in the schematic.

Session 3: Defining the PLD Equations

You can define the function of the PLD in a schematic either directly in the PLUSASM equation language or through a third-party PLD compiler (for example, a Xilinx ABEL file that produces a .PLD file equivalent to RCVR.PLD is shown in). This session demonstrates the creation of a simple PLD equation file using PLUSASM.

You can use the PLUSASM language to define the function of a PLD in a schematic in terms of Boolean equations. In this session you will learn how to develop the equation file, RCVR.PLD, for the PL22V10 PLD used in the UART schematic. The following functions are introduced in this session:

STEP 1: Define the Declaration Statements

STEP 2: Create the Boolean Equations

Step 1: Define the Declaration Statements

Display the contents of the rcvr.pld text file using an operating system editor such as vi. This file is located in $xilinx_tutorial / uart. The RCVR Boolean equation file is shown in the figure below.

The declarations section contains design and signal identification information. The PLUSASM keywords TITLE, AUTHOR, COMPANY, and DATE identify the design. The keyword CHIP identifies the PLD equation file name RCVR and PLD type PL22V10. The last two lines in declarations section form a sequential list of signal names and polarities for each pin on the PL22V10.

Step 2: Create the Boolean Equations

The keyword EQUATIONS identifies the beginning of the equations section. Following the EQUATIONS keyword are the Boolean equations written for each output signal used in the PL22V10.

For more detailed information regarding the syntax of PLUSASM equations and keywords, refer to the PLUSASM Language Reference in the *XEPLD Reference Manual*.

Exit your editor.

```
TITLE        UART Receiver Controller
AUTHOR       Applications
COMPANY      Xilinx
DATE         February 1993

CHIP    RCVR    PL22V10

; 1      2   3   4   5   6   7   8     9     10  11  12
x4clk  c0  c1  c2  c3  c4  c5  read  sdin  d0  nc  gnd

; 13  14  15   16     17      18  19     20     21     22
23  24 nc  nc  start bitclk byteclk par framing parity overun
ready nc  vcc

EQUATIONS

/start := /start * sdin
+ c5*/c4*c3*/c2*/c1*c0
; start goes high when sdin goes low
; and stays high until count=41.

bitclk := /c0 * /c1 * start
```

```
; bitclk pulses every 4 clock cycles
; to strobe deserializer.

ready := c5*/c4*c3*/c2*/c1*c0 * /parity * /framing * /overun
+ ready * /read
; ready goes high at count=41 if no errors
; and stays high until read.

byteclk := c5*/c4*/c3*/c2*c1*/c0 * /ready
; byteclk strobes output_reg at count=34
; only if ready not still active.


overun := c5*/c4*/c3*/c2*c1*/c0 * ready
+ overun * /read
; overrun error at count=34 if ready still
; active; stays on until read.


par := par * /sdin * bitclk * start
+ /par * sdin * bitclk * start
+ par * /bitclk * start
; accumulate parity on sdin on each bitclk;
; reset while start=0.


parity := c5*/c4*/c3*/c2*c1*/c0 * par
+ parity * /read
; parity error at count=34 if par odd (1);
; stays on until read.


framing := c5*/c4*c3*/c2*/c1*/c0 * /sdin
+ c5*/c4*c3*/c2*/c1*/c0 * /d0
+ framing * /read
; framing error at count=40 if either stop
; bit is low; stays on until read.
```

Figure 16-11 rcvr.pld File Contents

```
module rcvr

title ‘Control Logic and Error Detector for UART Receiver
Design
Xilinx EPLD Applications, Feb. 93’

        rcvr  device  ‘p22v10’;

“ Inputs

  x4clk             pin 1;           “ External clock (4x
                                     baud rate)

  c0,c1,c2,c3,c4,c5 pin 2,3,4,5,6,7; “ State counter outputs
                                     (from cntr6)

  read              pin 8;           “ Read enable (from
                                     cntr6,active-high)

  sdin              pin 9;           “ Serial data input
                                     (external)

  d0                pin 10;          “ Shift register LSB
                                     output

“ Outputs

  start             pin 15 istype ‘reg’; “ Start bit detector

  bitclk            pin 16 istype ‘reg’; “ Bit clock (to
                                         shifter)

  byteclk           pin 17 istype ‘reg’; “ Output data
                                         register clock

  par               pin 18 istype ‘reg’; “ Parity accumulator

  framing           pin 19 istype ‘reg’; “ Framing error
                                         output (external)

  parity            pin 20 istype ‘reg’; “ Parity error
                                         output (external)

  overun            pin 21 istype ‘reg’; “ Overrun error
                                         output (external)

  ready             pin 22 istype ‘reg’; “ Receiver ready
                                         output (external)

“ Variables
   count = [c5..c0]; “ c5 is MSB

Equations

!start := !start & sdin              “ Start goes high when
                                     sdin goes low;

        # (count == 41);             “ start stays high until
```

```
                                    count=41.

start.clk = x4clk;

bitclk := !c0 & !c1 & start;      “ Bitclk pulses every 4
                                    cycles.

bitclk.clk = x4clk;

ready := (count == 41) & !parity & !framing & !overun

       # ready & !read;             “ Ready goes high at
                                    count=41 if no errors

ready.clk = x4clk;                  “ and stays high until
                                    register read.

byteclk := (count == 34) & !ready;  “ Strobe data register
                                    at count=34

byteclk.clk = x4clk;                “ only if ready not still
                                    active.

overun := (count == 34) & ready     “ Overrun error at
                                    count=34 if ready still
                                    on;

       # overun & !read;            “ overun stays on until
                                    register read.

overun.clk = x4clk;

par := (par $ sdin) & bitclk & start

     # par & !bitclk & start;       “ Accumulate parity of
                                    sdin on each bitclk;

par.clk = x4clk;                    “ reset while start=0.

parity := (count == 34) & par       “ Parity error at
                                    count=34 if par odd (1);

      # parity & !read;             “ parity stays on until
                                    register read.

parity.clk = x4clk;

framing := (count == 40) & (!sdin # !d0)“ Framing error at
                                    count=40 if either

      # framing & !read;            “ stop bit low;

framing.clk = x4clk;                “ framing stays on until
                                    register read.
end
```

Figure 16-12 rcvr.abl File Contents

Session 4: Fitting the Design

Session 4 concentrates on the XEPLD Fitter. The following tasks are explained in this session.

STEP 1: Invoke the Fitter

STEP 2: View the Reports

Step 1: Invoke the Fitter

You can create a netlist from the schematic, process the PLD file, fit the design, and optionally create an Intel Hex programming file using a single command.

1. Select the icon for the UART design in the directory window.
2. Select **Open → pld_men2xnf8** from the pop-up menu.
3. A dialog box appears with options. Type **7354-12PC68** in the Part Type field. Select **OK**.
4. Select the xnf uart icon, which should be just above the original uart icon. Select **Open → pld_xemake** from the pop-up menu.
5. A second dialog box appears. To create an Intel HEX programming file, select the **File** option under Target and enter **uart.prg** in the text box that appears.
6. Because you are creating a programming file, you will also need a signature. Select the **Yes** option under **Signature** and enter **uart** in the text box that appears.
7. Do not change any other options. Select **OK** to start the Fitter.

For instructions on how to invoke the commands and options behind PLD_Men2XNF8 and PLD_XEMake, see the “Manual Translation” chapter.

Alternative Ways to Process PLDs

XEPLD can also read 20V8 or 22V10 PLD designs in the form of JEDEC standard programming maps produced by any third party PLD compiler.

Some third-party PLD compilers produce PALASM-compatible Boolean equation output files. Most PALASM equation syntax can be read directly by the PLUSASM assembler. For example, the logic for the PLD used in the UART design could be implemented using Xilinx ABEL syntax, as shown in, and compiled by Xilinx ABEL to produce a PLD formatted equation file.

For instructions on using JEDEC or PALASM files, see the “Manually Processing Your Design” chapter of this manual.

Step 2: View the Reports

The following extensions designate the reports produced by the PLD_XEMake command. These reports are in the /home/bclinton/mentor/uart directory.

.err	The Fitter Error report
.lga	The Assembler Log report
.res	The Resource report
.map	The Mapping report
.pin	The Pinlist report
.lgc	The Logic Optimizer report
.par	The Partitioner report

You can view the reports that the Fitter generates using your favorite editor.

1. First, view the Fitter Error and Assembler Log reports if error messages appeared on the screen during fitting.

   PLUSASM stores errors and warnings in a report file called rcvr.err. PLUSASM also produces a detailed report of its results in a file named rcvr.lga (the assembler log).

2. Next, view the Resource report, uart.res.

   The Resource report lists the amount of resources that were used to implement the design. This report contains the total number of macrocell and Function Block resources and input/output (I/O) pins used on the target device. These totals are subtracted from the total resources of the device to give the amount of remaining resources available to the designer.

3. Next, view the Mapping report, uart.map.

   The Mapping report lists each Function Block in the device and details which component outputs were mapped to macrocells of each Function Block. The Mapping report is used primarily for design debugging and to assist manual mapping.

4. Next, view the Pinlist report, uart.pin.

   The Pinlist report provides the designer with chip pin placement information. For each pin on the package, the Pinlist report indicates the operation of the pin as used in the design and the signal in your design appearing on the pin. The signal names in the Pinlist report are the names you placed on the nets connected to PAD symbols in your schematic.

5. Next, view the Logic Optimizer report, uart.lgc.

   The Logic Optimizer report lists which inputs have been collapsed into their fanouts and which outputs have been optimized.

6. Finally, view the Partitioner report, uart.par.

   The Partitioner report provides cross-reference tables showing the allocation of all Function Block resources. This report provides detailed information for optimizing your design and manipulating chip resources.

Note: When you are using schematic capture, XEPLD uses the names passed to it from Mentor Graphics. Understanding these reports with Mentor Graphics-assigned names can be very difficult. For this reason, designers are encouraged to provide unique names to symbols and nets.

Session 5: Simulating the Design

Session 5 gives an example of one way to simulate the UART design. The following tasks are explained in this session:

STEP 1: Prepare Your Input Vectors

STEP 2: Run the Simulation and View the Results

You can end the tutorial here if you are familiar with QuickSim.

Step 1: Prepare Your Input Vectors

Before you run the simulation, you should prepare a list of inputs in a do file. You can enter QuickSim and specify inputs one at a time, but it is often easier and faster to put them in a do file.

A do file, uart.do, is provided for you in /home/bclinton/mentor directory. The following figure is an abbreviated listing of its contents — most clock input vectors have been removed.

You can view the entire do file in any ASCII text editor.

```
ADD BUS    DOUT DOUT(7) DOUT(6) DOUT(5) DOUT(4) DOUT(3)
DOUT(2) DOUT(1) DOUT(0)
ADD LIST   //PRLD
ADD LIST   X4CLK
ADD LIST   RD
ADD LIST   CS
ADD LIST   SDIN
ADD LIST   START
ADD LIST   BITCLK
ADD LIST   BYTECLK
ADD LIST   DOUT
ADD LIST   READY
ADD LIST   OVERUN
ADD LIST   PARITY
ADD LIST   FRAMING
ADD TRACE  //PRLD
ADD TRACE  X4CLK
ADD TRACE  RD
ADD TRACE  CS
ADD TRACE  SDIN
ADD TRACE  START
ADD TRACE  BITCLK
ADD TRACE  BYTECLK
ADD TRACE  DOUT
ADD TRACE  READY
ADD TRACE  OVERUN
ADD TRACE  PARITY
ADD TRACE  FRAMING
DELETE FORCE -all


FORCE //PRLD 1 0 -F
FORCE //PRLD 0 50 -F
FORCE X4CLK 1 0 -F
FORCE X4CLK 0 150 -F
FORCE X4CLK 1 200 -F
FORCE X4CLK 0 250 -F
FORCE X4CLK 1 300 -F
FORCE X4CLK 0 350 -F
FORCE X4CLK 1 400 -F
FORCE X4CLK 0 450 -F
FORCE X4CLK 1 500 -F
...
FORCE X4CLK 1 10700 -F
FORCE X4CLK 0 10750 -F

FORCE RD 1 0 -F
FORCE RD 0 4650 -F
FORCE RD 1 5600 -F
FORCE CS 1 0 -F
FORCE CS 0 4800 -F
FORCE CS 1 5600 -F
FORCE SDIN 1 0 -F
FORCE SDIN 0 350 -F
FORCE SDIN 1 750 -F
FORCE SDIN 0 1550 -F
FORCE SDIN 1 1950 -F
FORCE SDIN 0 3150 -F
FORCE SDIN 1 3550 -F
FORCE SDIN 0 5950 -F
FORCE SDIN 1 6350 -F
FORCE SDIN 0 7550 -F
FORCE SDIN 1 8750 -F
FORCE SDIN 0 9550 -F
FORCE SDIN 1 9950 -F

RUN 11000
```

Figure 16-13 The uart.do File

The commands in the uart.do file perform the following actions:

- The ADD BUS command groups the DOUT signal into a bus.
- The ADD LIST commands add all the input and output signals and buses to the list file, which you will view later.
- The ADD TRACE commands add all the input and output signals and buses to the trace file, which you will view later.
- The DELETE FORCE -all command deletes any FORCE commands that might still be in effect from the last simulation.
- The FORCE commands are input vectors: they specify at which times each input changes value from 0 to 1 or from 1 to 0. First the

X4CLK vectors are listed, then the RD vectors, the CS vectors, and finally the SDIN vectors.

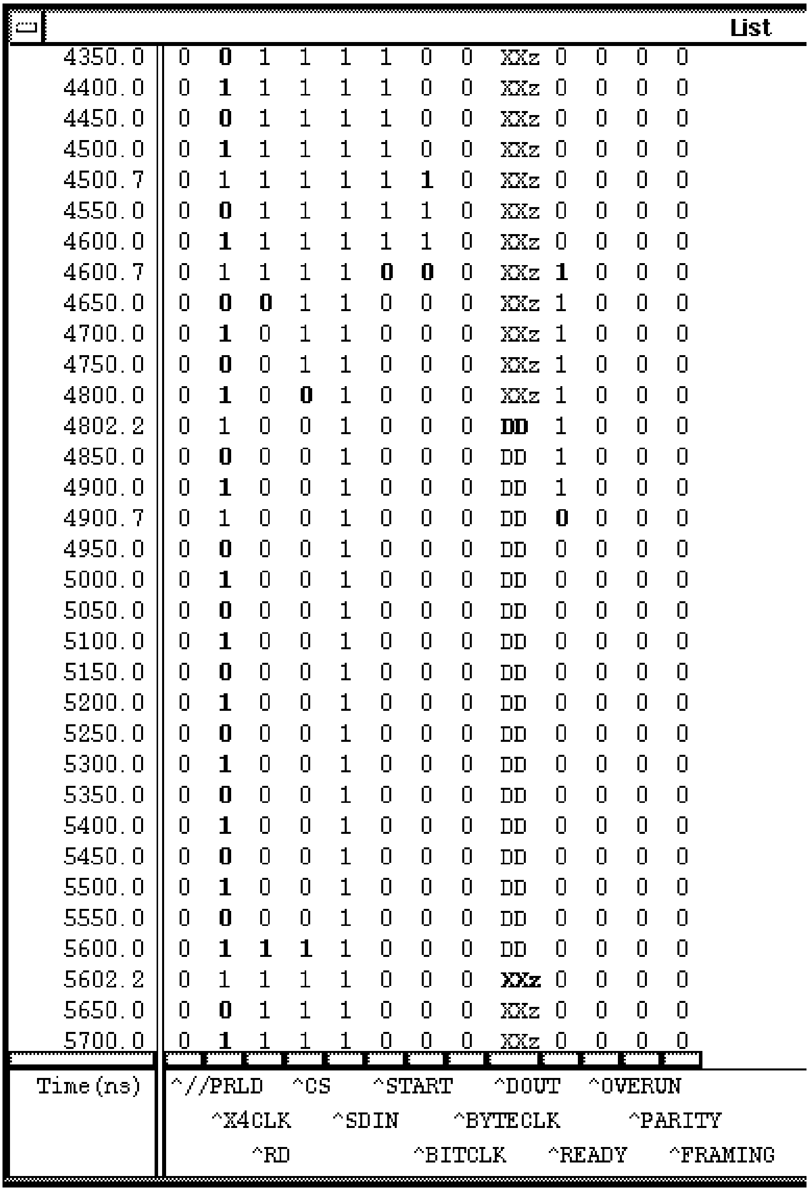
- The RUN command specifies how long the simulation will run, which is 11000 nsec in this case.

Step 2: Run the Simulation and View the Results

To run the simulation and view the results, follow these steps:

1. Select the uart icon in the Directory window of the Design Manager, then select the **`Open`** → **`pld_timsim8`** command from the pop-up menu.
2. Select the **`Auto Generate`** option.
3. Select **`Yes`** under the Run QuickSim? option.
4. Select the **`OK`** button to start the simulation.

   A window appears and displays messages about the process of making a simulation netlist file. If this process runs successfully, the QuickSim window opens.
5. Expand the QuickSim window by clicking on the button in the top right corner that contains the larger square.
6. Move the mouse to the large display area, then select the **`Force`** → **`From File`** command from the pop-up menu.
7. Type **`uart.do`** in the prompt window. Select **`OK`**.
8. In a few minutes, list and trace files appear. Iconize the trace file by clicking on the button in the top right corner that contains the smaller square.
9. Expand the list file view by clicking on the button in the top right corner that contains the larger square. The following figure shows what the list file should look like.
10. Use the PgUp and PgDn keys to scroll through the listing. The Y-axis displays the times, with zero at the top, and the X-axis displays the signal names. Note that whenever the value of a signal changes, the new value is displayed in green.
11. Iconize the list file and click twice on the trace file icon. Expand the trace file window.



List

```
   4350.0 || 0  0  1  1  1  1  0  0  XXz 0  0  0  0
   4400.0 || 0  1  1  1  1  1  0  0  XXz 0  0  0  0
   4450.0 || 0  0  1  1  1  1  0  0  XXz 0  0  0  0
   4500.0 || 0  1  1  1  1  1  0  0  XXz 0  0  0  0
   4500.7 || 0  1  1  1  1  1  1  0  XXz 0  0  0  0
   4550.0 || 0  0  1  1  1  1  1  0  XXz 0  0  0  0
   4600.0 || 0  1  1  1  1  1  1  0  XXz 0  0  0  0
   4600.7 || 0  1  1  1  1  0  0  0  XXz 1  0  0  0
   4650.0 || 0  0  0  1  1  0  0  0  XXz 1  0  0  0
   4700.0 || 0  1  0  1  1  0  0  0  XXz 1  0  0  0
   4750.0 || 0  0  0  1  1  0  0  0  XXz 1  0  0  0
   4800.0 || 0  1  0  0  1  0  0  0  XXz 1  0  0  0
   4802.2 || 0  1  0  0  1  0  0  0  DD  1  0  0  0
   4850.0 || 0  0  0  0  1  0  0  0  DD  1  0  0  0
   4900.0 || 0  1  0  0  1  0  0  0  DD  1  0  0  0
   4900.7 || 0  1  0  0  1  0  0  0  DD  0  0  0  0
   4950.0 || 0  0  0  0  1  0  0  0  DD  0  0  0  0
   5000.0 || 0  1  0  0  1  0  0  0  DD  0  0  0  0
   5050.0 || 0  0  0  0  1  0  0  0  DD  0  0  0  0
   5100.0 || 0  1  0  0  1  0  0  0  DD  0  0  0  0
   5150.0 || 0  0  0  0  1  0  0  0  DD  0  0  0  0
   5200.0 || 0  1  0  0  1  0  0  0  DD  0  0  0  0
   5250.0 || 0  0  0  0  1  0  0  0  DD  0  0  0  0
   5300.0 || 0  1  0  0  1  0  0  0  DD  0  0  0  0
   5350.0 || 0  0  0  0  1  0  0  0  DD  0  0  0  0
   5400.0 || 0  1  0  0  1  0  0  0  DD  0  0  0  0
   5450.0 || 0  0  0  0  1  0  0  0  DD  0  0  0  0
   5500.0 || 0  1  0  0  1  0  0  0  DD  0  0  0  0
   5550.0 || 0  0  0  0  1  0  0  0  DD  0  0  0  0
   5600.0 || 0  1  1  1  1  0  0  0  DD  0  0  0  0
   5602.2 || 0  1  1  1  1  0  0  0  XXz 0  0  0  0
   5650.0 || 0  0  1  1  1  0  0  0  XXz 0  0  0  0
   5700.0 || 0  1  1  1  1  0  0  0  XXz 0  0  0  0

 Time(ns) || ^//PRLD  ^CS   ^START    ^DOUT  ^OVERUN
          ||    ^X4CLK   ^SDIN    ^BYTECLK     ^PARITY
          ||       ^RD         ^BITCLK    ^READY    ^FRAMING
```

Figure 16-14   The List File

12. To see the waveforms from the beginning of the simulation, select the **View → All** command from the banner menus. The following figure shows what the trace file should look like.

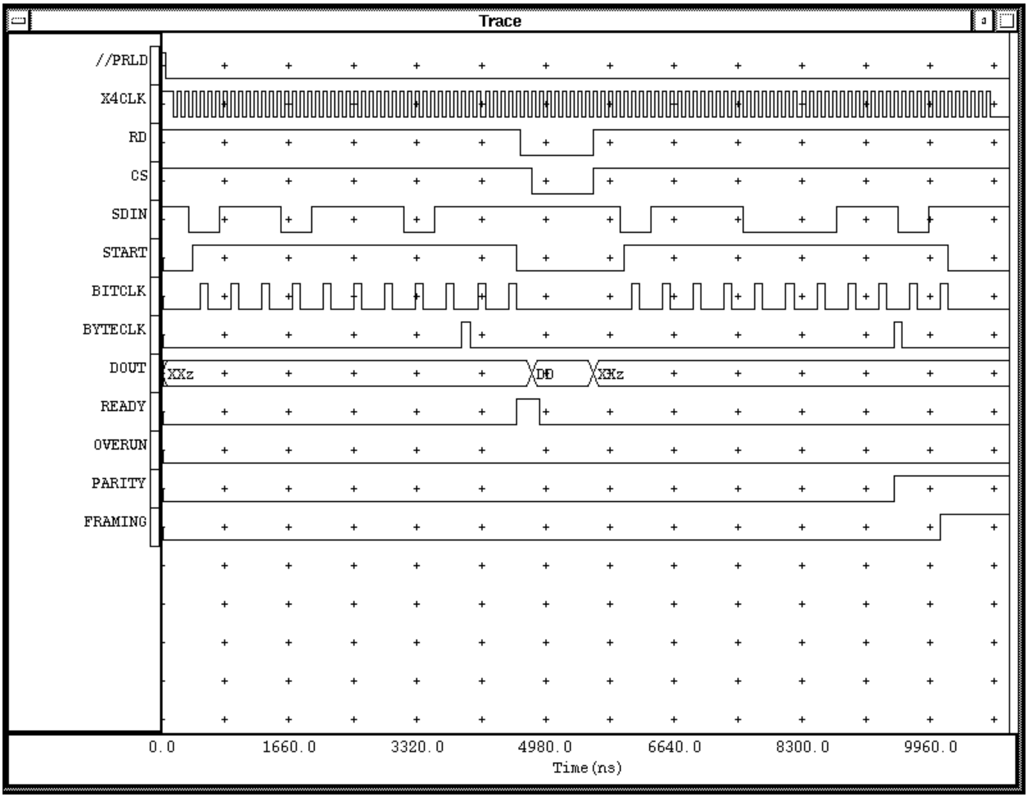


Figure 16-15 The Trace File

Note that between about 5000 and 5800 nanoseconds the output data bus is active. At about 9600 nanoseconds a parity error occurs, and at about 10200 nanoseconds a framing error occurs. (These errors were introduced deliberately to test whether the design could catch them.)

To quit QuickSim, move the mouse cursor to the square in the upper left corner of the Mentor Graphics window (which is inside the operating system window), press and hold the left mouse button, and select **Close** from the displayed menu. The Exit QuickSim dialog box appears. Select the **Without Saving** button followed by the **OK** button.

Session 6: Completely Schematic-Based Designs and Functional Simulation

Session 6 shows a version of the UART design in which the PL22V10 has been replaced with a lower-level schematic, which represents the RCVR function and has the same logic as the rcvr.pld file. This session also outlines the steps for functional simulation, which you can only perform for a purely schematic design.

STEP 1: Change the Working Directory

STEP 2: Create a Custom Symbol

STEP 2: Place the Custom Symbol in the Schematic

STEP 3: Create the Lower-Level Schematic

STEP 4: Prepare for Functional Simulation

STEP 5: Quit Mentor Graphics

Step 1: Change the Working Directory

1. **MGC → Location Map → Set Working Directory**
2. Type **$xilinx_tutorial/uarttop** in dialog box at bottom of window.
3. Select **OK**.

Step 2: Create a Custom Symbol

You can create your own symbols to place on the schematic. Under these symbols, you can build other schematics. Creating a design with multiple levels allows you to focus on specific parts of the design rather than trying to understand the entire design all at once.

The next subsections explain how to create a custom symbol.

Open a Symbol Window

To open a symbol window, follow these steps:

1. Double click on the PLD_DA (Design Architect) tool to open it.
2. Expand the Design Architect window by clicking on the button in the top right corner that contains the larger square.
3. Click on the **Open Symbol** box in the palette window to open a symbol window. If this box is not visible, select the **Session** button in the palette window to display it.
4. A prompt box appears. Type the following in the Component Name field:

   $xilinx_tutorial/uarttop/rcvrsub2

5. Select the **OK** button. The symbol window appears.
6. Expand the symbol window by clicking on the button in the top right corner that contains the larger square.
7. Zoom out so that the view in the symbol window is about 25 grid units high.

Draw the Rectangle

To draw the rectangle of the symbol, follow these steps:

1. Select the **Add Rectangle** box in the Palette window. You will make a rectangle that is 12 grid units wide and 22 grid units high. Be sure that enough grid units show in the window.
2. Position the cursor at the top of the symbol window a little bit left of center and press the left mouse button.
3. While holding down the left mouse button, move the cursor 12 grid units to the right and 22 grid units down. Release the mouse button.

Add the Pins

To add the pins, follow these steps:

1. Select **Add Pin** from the Palette window.
2. In the dialog box that comes up, select the **<>-Name** option under Name Placement, **IN** for the pin type, and the **<>-|** option under Pin Placement.

3. Type the following names in the Pin Name(s) fields. Do not press return after entering a name. Note that each time you type a name, another pin name box appears.

   X4CLK
   C0
   C1
   C2
   C3
   C4
   C5
   READ
   SDIN
   D0

4. Select the **`OK`** button.
5. The dialog box disappears and the mouse changes to a star. Move the mouse one grid unit to the left side of the rectangle and two grid units below the top of the rectangle. Click with the left mouse button.
6. Move the mouse two grid units down. Click with the left mouse button again to place the second pin.
7. Continue moving the mouse down two grid units and clicking to place pins until all the pins are placed.
8. Select **`Add Pin`** again.
9. In the dialog box that comes up, select the **`<>-Name`** option under Name Placement, **`IXO`** for the pin type, and the **`|-<>`** option under Pin Placement.
10. Type the following names in the Pin Name(s) fields:

    READY
    OVERUN
    PARITY
    FRAMING
    BYTECLK
    BITCLK
    START

11. Select the **`OK`** button.

12. The dialog box disappears and the mouse changes to a star. Move the mouse one grid unit to the right side of the rectangle and four grid units below the top of the rectangle. Click with the left mouse button.
13. Move the mouse two grid units down. Click with the left mouse button again to place the second pin.
14. Continue moving the mouse down two grid units and clicking to place pins until all the pins are placed. Move down four units between the FRAMING and BYTECLK pins.

Lengthening the Pins

The pins are each one grid unit long. To make them each two grid units long, follow these steps:

1. Select all of the pins on the left side of the symbol, being careful NOT to select the lines connected to the pins.
2. Move the pins one grid unit to the left using the **`Move`** command on the pop-up menu. Press F2.
3. Move the pin names to the right one grid unit. Use F1 to select each pin name (positioning the mouse in the lower left corner of the text before pressing F1 works best). Press F2.
4. Select all of the lines, and select **`Delete`** on the pop-up menu.
5. Select the **`Two Point Line`** command from the pop-up menu. Move the mouse to the X4CLK pin. Press and hold the left mouse button as you drag the mouse to the rectangle edge. The resulting line should be two grid units long.
6. Use the **`Copy`** command on the pop-up menu to copy the line you just drew. Place the copy at the C0 pin.
7. Continue copying lines until all the pins on the left side of the symbol have lines.
8. Repeat the above steps to lengthen the pins on the right side of the symbol.
9. Select each of the pin names using the F1 key. Then select the **`Change Height → 1.0`** command from the pop-up menu.

Add the Symbol Name

To name the symbol and reference the schematic that will be under it, follow these steps:

1. Press F2 to deselect everything, then select the rectangle.
2. Select **`Properties → Add`**.
3. In the dialog box that comes up, type **`REF`** in the Property Name box and **`rcvrsub`** in the Property Value box. Select **`OK`**.
4. Move the rcvrsub text to the top of the rectangle and click to place the text.
5. Deselect everything using F2.
6. Select the rcvrsub text using F1.
7. Select the **`Change Height → 2.0`** command.
8. Use the **`Move`** command to adjust the rcvrsub text position.

Check the Symbol

Select **`Check → With Defaults`** to verify that your symbol does not violate any symbol rules.

Save the Symbol

To save the symbol, you can do one of the following:

- Select the **`Save → Symbol`** command (File menu) to save to the name rcvrsub2, then use the rcvrsub symbol supplied by Xilinx in the next step.
- Select the **`Save → Symbol As...`** command (File menu) and save to the following Component Name:

  $xilinx_tutorial / uarttop / rcvrsub

  This overwrites the rcvrsub symbol supplied by Xilinx and allows you to use the symbol you created in the next step.

When you are finished, the RCVRSUB symbol should appear as in the following figure.

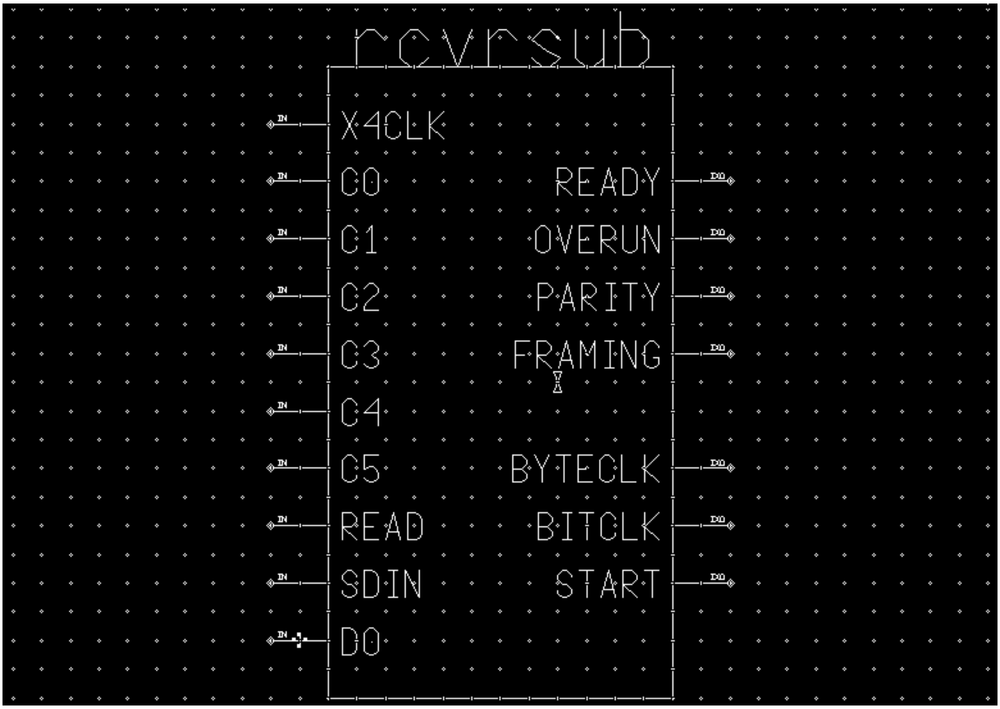


Figure 16-16 The RCVRSUB Custom Symbol

Step 3: Place the Custom Symbol in the Schematic

Next, you must place the new symbol in the top-level schematic.

To place the RCVRSUB symbol in the UART schematic, follow these steps:

1. Close the symbol window. Move the mouse cursor to the square in the upper left corner of the symbol window, press and hold the left mouse button, and select **`Close`** from the displayed menu.
2. Select **`File`** → **`Open`** → **`Sheet`**.
3. A dialog box appears. Type **`$xilinx_tutorial/uart/uart`**.
4. Select the **`OK`** button or press Return.

5. Click on the square in the upper right corner of the schematic window to expand it. Press Shift-F8 to view the entire sheet at once.
6. Use the **`Delete`** command to delete the PL22V10.
7. Use the **`Move`** command to move the ends of the nets that were connected to the right side of the PL22V10 two grid units to the right.
8. Select the **`Choose Symbol`** icon in the palette and select **`rcvrsub`** from the list. Place the new symbol in the space left by the PL22V10 symbol. The wires and the RCVRSUB pins should line up.
9. Select the **`Properties → Add`** command and name the RCVRSUB symbol controller.
10. Select the **`File → Save Sheet As...`** command and type the following in the Component Name field:

    $xilinx_tutorial / uarttop / uarttop2

11. Click on the **`OK`** button.

After you have added the RCVRSUB symbol, the schematic should appear as shown in the following figure.

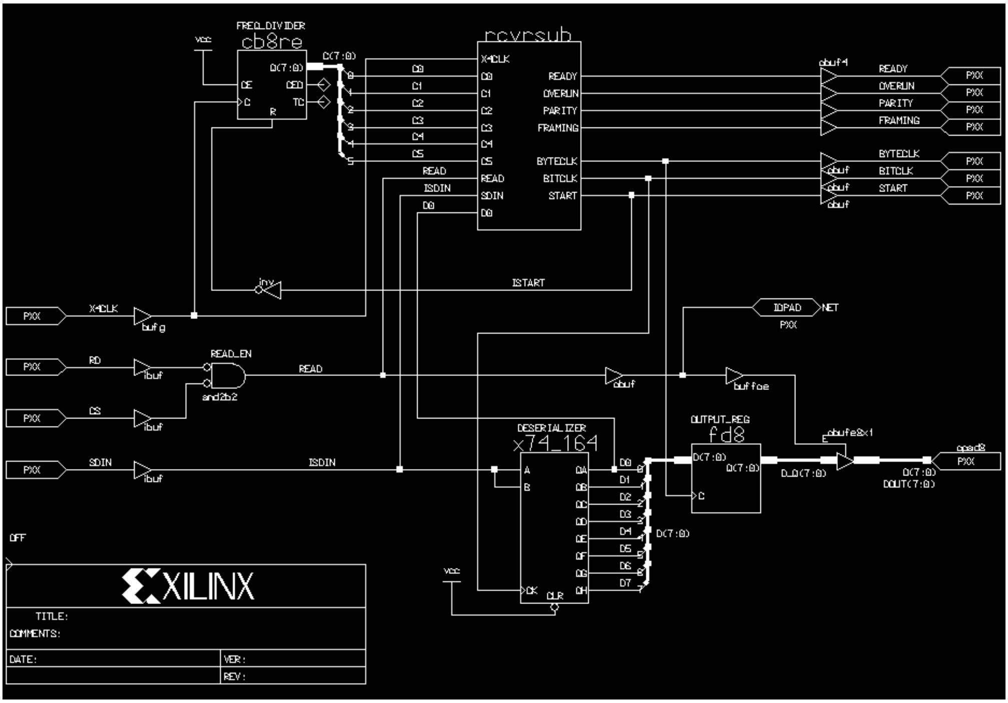


Figure 16-17 The UART Schematic with the RCVRSUB Symbol

Step 4: Create the Lower-Level Schematic

The next step is to create the lower-level schematic. To save you time going through this tutorial, we have created this file for you.

To view the RCVRSUB schematic, follow these steps:

1. Select the rcvrsub symbol in the uarttop schematic.
2. Select the **Open → Down** command and select **schematic sheet1** from the list. Click **OK**. A schematic sheet named RCVRSUB opens.
3. Click on the square in the upper right corner of the schematic window to expand it. Press Shift-F8 to view the entire sheet at once.

4. The schematic is shown in the figure below. The only components it contains are INV, OR2, NOR2, OR3, FD, and AND*n*, where *n* is a number between 2 and 9 that indicates the number of inputs.

   The wire names match inputs and outputs in the lower-level schematic to the pins in the RCVRSUB symbol.

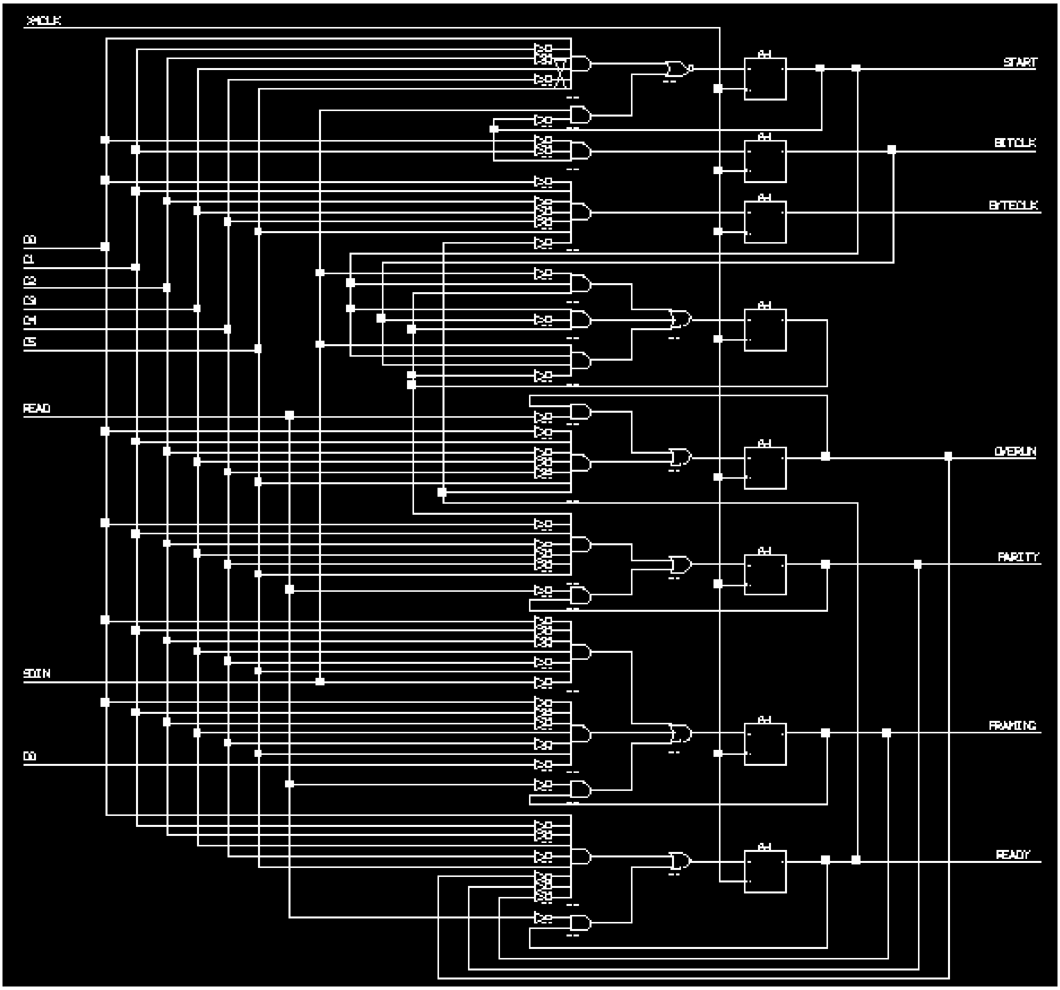


Figure 16-18   The RCVRSUB Schematic

5. Close Design Architect. Move the mouse cursor to the square in the upper left corner of the Design Architect window, press and hold the left mouse button, and select **`Close`** from the displayed menu.

Step 5: Run the Simulation and View the Results

To run the simulation and view the results, follow these steps:

1. Select the icon for the uarttop design in the directory window.
2. Select **`Open`** → **`pld_men2xnf8`** from the pop-up menu.
3. A dialog box appears with options. Type **`7354-12PC68`** in the Part Type field. Select **`OK`**.
4. Select the uarttop icon in the Directory window of the Design Manager, then select the **`Open`** → **`pld_fncsim8`** command from the pop-up menu.
5. Select the **`Use Original`** option.
6. Select **`Yes`** under the Run QuickSim? option.
7. Select the **`OK`** button to start the simulation.

   A window appears and displays messages about the process of making a simulation netlist file. If this process runs successfully, the QuickSim window opens.
8. Move the mouse to the large display area, then select the **`Force`** → **`From File`** command from the pop-up menu.
9. Type **`uarttop.do`** in the prompt window. Select **`OK`**.
10. In a few minutes, list and trace files appear. Iconize the trace file by clicking on the button in the top right corner that contains the smaller square.
11. Expand the list file view by clicking on the button in the top right corner that contains the larger square.
12. Use the PgUp and PgDn keys to scroll through the listing. The Y-axis displays the times, with zero at the top, and the X-axis displays the signal names. Note that whenever the value of a signal changes, the new value is displayed in green.

13. Iconize the list file and click twice on the trace file icon. Expand the trace file window.
14. To see the waveforms from the beginning of the simulation, select the **`View → All`** command from the banner menus. The results should be the same as for timing simulation; the figure labeled Trace File above shows what the trace file should look like.
15. To quit QuickSim, move the mouse cursor to the square in the upper left corner of the Mentor Graphics window (which is inside the operating system window), press and hold the left mouse button, and select **`Close`** from the displayed menu. The Exit QuickSim dialog box appears. Select the **`Without Saving`** button followed by the **`OK`** button.

Step 6: Quit Mentor Graphics

To quit, move the mouse cursor to the square in the upper left corner of the Mentor Graphics window (which is inside the operating system window), press and hold the left mouse button, and select **`Close`** from the displayed menu.

You have now finished the XEPLD Mentor Graphics Tutorial.

Mentor Graphics Interface/ Tutorial Guide

Error Messages

Appendix A

Error Messages

This section lists warning and error messages, and possible recovery solutions.

EDIF2XNF

In the following error messages, variables are preceded by a “$” sign and are written in italics; however, EDIF2XNF displays information specific to your software (for example, file names and port names) on–screen.

> `Error 1: Cell $`*`cellref`*` not found in library $library for cellRef on instance $`*`instance`*` in cell $`*`cell`*`. Explicit libraryRef may be missing. Bad EDIF input file!`

> `Error 2: library reference name $`*`library`*` not found for design $`*`design`*`. Bad EDIF input file!`

> `Error 3: Port name $`*`port`*` not found on external library primitive for cell $`*`cell`*`.`

EDIF2XNF uses EDIF data files that describe the known set of Xilinx primitives for a technology. These files are usually located in **`$XACT/data/edif2000`**, **`$XACT/data/edif3000`**, or **`$XACT/data/edif4000`**. They contain a stripped–down version of the EDIF netlist extracted from Xilinx’s set of golden symbols. The netlist contains a primitive that is not recognized. An EDIF primitive is defined as a cell with no contents record. Exceptions to this rule are user–defined primitives created using the **`FILE=<fn>[.xnf]`** property or **`DEF=HM`** property (user–defined hard macro, XC4000 only).

> `Error 4: No cellRef for instance $`*`instance`*` in cell $`*`cell`*`. Bad EDIF input file!`

```
Error 5: No direction/pintype for port:
$instance_port.
```

```
Error 6: EDIF data $file not found in directory
$directory. Cell $cell is not a recognized Xilinx
primitive component.
```

There is no CONTENTS record found in this cell. Refer to the explanation for Error 3.

```
Error 7: Port $port not mapped in map file for
primitive cell $cell.
```

The Map File feature is not supported.

```
Error 8: Library not found for libraryRef $library
on instance $instance in cell $cell. Bad EDIF input
file!
```

```
Error 9: Output directory $directory does not exist.
```

```
Error 10: No design records found in $edif_input_file.
```

```
Error 11: Part type not specified for design
$design in the file $file.
```

```
Error 12: Part type $part is invalid for design
$design in file $file.
```

Specify the part type with the EDIF2XNF command-line option, -p.

```
Error 13: Unknown direction for port $port.
```

```
Error 14: PortRef $port not found for instance
$instance on net $net in cell $cell. Bad EDIF input
file!
```

Net arrays might not contain PortRef references to PortBundles, because EDIF does not have a method of specifying how to connect the bits in the Net Array to the PortBundle, (low order / high order).

Workaround: Label the Net using the same convention as the PortBundle it is connected to and do not mix Bundles and Arrays. For example:

Okay: `Port A<5,2> - Net B<3,1> - Port C<5,1>`
Not okay: `Port A<5,2> - Net B<0:1> - Port C<5,1>`

Explanation: Some schematic-capture programs attempt to use EDIF in an illegal way. Connecting bundles to arrays is not legal within EDIF, because by definition, nets within a bundle can be in any order. The program reading the EDIF file does not recognize the order, and cannot make any assumptions.

```
Error 15: Instance $instance not found for
instanceref in joined record on net $net in cell
$cell. Bad EDIF input file!

Error 16: Cell $cell not found in library $library
for cellRef on instance $instance in cell $cell. Bad
EDIF input file!

Error 17: Cell $cell not found for design $design.

Error 18: Out of memory.

Error 19: No direction for port $port in external
library primitive $cell.

Error 20: PortRef $port not found within
portBundle $portBundle for instance $instance in
joined record for net $net in cell $cell. Bad EDIF
input file!

Error 21: PortBundle $portBundle not found for
instance $instance in joined record for net $net in
cell $cell. Bad EDIF input file!

Error 22: Could not open $file for read.

Error 23: Could not open $file for write.

Error 30: Cell $cell not found in Verilog map file.
```

The EDIF2VERILOG program requires a map file to map XNF pin names to Verilog pin names.

```
Error 31: Could not checkout license for $EDIF
version $VERSION EDIF2VERILOG requires the
correct Xilinx license server.

Error 32: LCANET %s record not valid, must be 4
or 5.

Error 33: XACT environment variable not set or
lib_path arg missing.
```

Gen_Sch8

This section lists Gen_Sch8 warning and error messages.

```
WARNING 48: Aka file read error: <aka_error>.
```

Something is wrong in the AKA file. Check the file.

```
WARNING 86: Unknown command-line option flag
<flag> ignored.
```

An unexpected option was specified.

```
WARNING 87: Extra command-line argument <arg>
ignored.
```

You specified too many arguments in the command line.

```
WARNING 220: Can’t open file <file_name>.
```

```
Possible reasons:
File does not exist in current directory.
You do not have read privilege to the file.
File is locked by another process.
```

```
Error 220: Can’t open file <input_file_name>.
Possible reasons:
File does not exist in current directory.
You do not have read privilege to the file.
File is locked by another process.
```

```
Error 230: <net_reader_message>.
Aborting due to errors in <xnf_filename> xnf
file.
```

XNF file is corrupt; regenerate XNF file.

```
Error 283: Missing aka file name.
The -a command-line option flag must be followed
by an aka file name.
```

```
Error 285: Missing output file name.
The -o command-line option flag must be followed
by a new file name.
```

```
Error 286: LCA environment variable undefined.
```

You need to set an LCA environment variable, pointing to the location of DS344 software and libraries.

```
Error 287: Missing sheet number.
The -s command line option flag must be followed
by a positive non-zero integer.
```

```
Error 327: Could not find <symbol_name> in
<library_pathname> library.
```

XNF file is corrupt; regenerate XNF file.

```
Error 384: Check Sheet failed.
```

Invoke Design Architect on the newly generated schematic. Perform a check sheet on the schematic in Design Architect.

```
Error 396: Could not find pin <pin_name> on
symbol <symbol_name>.
```

XNF file is corrupt; regenerate XNF file.

Gen_Sym8

This section lists Gen_Sym8 warning and error messages.

```
WARNING 86: Unknown command-line option flag
<flag> ignored.
```

An unexpected option was specified.

```
WARNING 87: Extra command-line argument <arg>
ignored.
```

You specified too many arguments in the command line.

```
WARNING 220: Can’t open file <file_name>.
Possible reasons:
File does not exist in current directory.
You do not have read privilege to the file.
File is locked by another process.
```

```
Error 220: Can’t open file <input_file_name>.
Possible reasons:
File does not exist in current directory.
You do not have read privilege to the file.
File is locked by another process.
```

```
Error 230: <net_reader_message>.
Aborting due to errors in <xnf_filename> xnf
file.
```

XNF file is corrupt; regenerate XNF file.

```
Error 250: Error while writing DA script to disk.
Some information may be found in file
<temp_filename>.
Check for disk full condition.
```

Check to see that you have write permission.

```
Error 281: Unable to generate <output_pathname>
file name - too long.
```

Use the -o option and enter a shorter pathname.

```
Error 285: Missing output file name.
The -o command-line option flag must be followed
by a new file name.
```

```
Error 287: Unable to open temporary work file
<temp_filename>.
```

Make sure file exists in current directory; you have read privilege to the file; and that the file is not locked by another process.

```
Error 289: Could find neither <design_name>.xnf
nor <design_name>.xsf file for input.
```

Make sure XNF or XSF file exists in your working directory.

PLD_DVE_BA

This section lists PLD_DVE_BA warning and error messages.

```
WARNING: mba_file_name not specified on command
line.
```

```
WARNING: component_name not found in current
directory.
```

```
WARNING: mba_file_name not found in current
directory.
```

```
Error: Cannot determine value of $LCA.
```

You need to set an LCA environment variable, pointing to the location of DS344 software and libraries for PLD_DVE_BA to work.

PLD_DVE

This section lists PLD_DVE warning and error messages.

```
WARNING: lca_technology not specified on command
line. Please enter lca technology for this
component (xc2000, xc3000, or xc4000):
```

```
Error: Invalid lca_technology type specified.
```

You must specify a valid XC library, either XC2000, XC3000, or XC4000.

```
Error: Cannot determine value of $LCA.
```

You need to set an LCA environment variable, pointing to the location of DS344 software and libraries for PLD_DVE_BA to work.

PLD_DVE_SIM

This section lists PLD_DVE_SIM warning and error messages.

```
WARNING: lca_technology not specified on command
line. Please enter lca technology for this
component (xc2000, xc3000, or xc4000):
```

```
WARNING: component_name not found in current
directory.
```

```
Error: Invalid lca_technology type specified.
```

You must specify a valid XC library, either XC2000, XC3000, or XC4000.

```
Error: Cannot determine value of $LCA.
```

You must specify a valid XC library, either XC2000, XC3000, or XC4000.

UNAKAXNF

This section lists UNAKAXNF warning and error messages.

```
WARNING 43: Invalid part type <part> ignored.
```

You must have a valid part type in the XNF file.

```
WARNING 48: Aka file read error: <aka_error>.
```

Something is wrong in the AKA file. Check the file.

```
WARNING 86: Unknown command–line option flag -
<option_flag> ignored.
```

An unexpected option was specified.

```
WARNING 87: Extra command–line argument <arg>
ignored.
```

You specified too many arguments in the command line.

```
Error 220: Can’t open file <input_file_name>.
  Possible reasons:
  File does not exist in current
  directory.
  You do not have read privilege to
  the file.
  File is locked by another process.
```

```
WARNING 221: Can’t open file <input_file_name>.
  Possible reasons:
  File does not exist in current
  directory.
  You do not have read privilege to
  the file.
  File is locked by another process.
  If map2lca was run with the -a
  option, you can ignore this
  warning.
```

```
Error 230: Aborting due to errors in
<xnf_file_name> netlist file.
```

Your XNF file is corrupt. The program aborted, because the XNF file contains too many errors. Check your XNF file.

```
Error 250: Error while writing XNF information to
disk. Some information may be found in file
<temp_file_name>. Check for disk–full condition.
```

Either your disk is full, or you do not have write permission for the directory.

```
Error 283: Missing aka file name.
```

The -a command-line option must be followed by an AKA file name.

```
Error 285: Missing output file name.
```

The -o command-line option must be followed by a new file name.

```
Error 287: Unable to open temporary work file
<temp_file_name>.
```

Either your disk is full, or you do not have write permission for the directory.

XBLXGS

This section lists XBLXGS warning and error messages.

```
WARNING 86: Unknown command-line option flag
<flag> ignored.
```

An unexpected option was specified.

```
WARNING 87: Extra command-line argument <arg>
ignored.
```

You specified too many arguments in the command line.

```
Error 11: Bottom-level symbol already there:
<sym_name>.
```

XGS file is corrupt; rerun X-BLOX to generate new XGS file.

```
Error 12: FILE property missing: <sym_name>.
```

FILE property was expected, but not found. Add FILE property to <sym_name> in your schematic.

```
Error 13: No PIN property found on symbol pin:
<symbol_name>.
```

PIN property was expected, but not found. Add PIN property to <symbol_name> in your schematic.

```
Error 14: Incomplete BUSSIGS record: <record>.
```

XGS file is corrupt; rerun X-BLOX to generate new XGS file.

```
Error 15: No INIT property found on symbol:
<symbol_name>.
```

INIT property was expected, but not found. Add INIT property to <symbol_name> in your schematic.

```
Error 16: Modified net width mismatch found on
net <net_name>.
```

XGS file is corrupt; rerun X-BLOX to generate new XGS file.

```
Error 17: Modified pin width mismatch found on
pin <pin_name>.
```

XGS file is corrupt; rerun X-BLOX to generate new XGS file.

```
Error 220: Can’t open file <input_file_name>.
Possible reasons:
File does not exist in current directory.
You do not have read privilege to the file.
File is locked by another process.
```

```
Error 230: <net_reader_message>.
Aborting due to errors in <xgs_filename> xgs
file.
```

```
Error 285: Missing output file name.
The -o command-line option flag must be followed
by a new file name.
```

XNFBA

This section lists XNFBA warning and error messages.

```
WARNING 202: The two input XNF files are
identical.
```

No operation is performed. The file names specified with the -a and -b options are the same. Check your command line and try again.

```
WARNING 203: Non–primitive <symbol name> is found
(TYPE=<type name>) in <xnf file>.
```

Run XNFMERGE to flatten the design, before routing it. Flattening a design before routing it can improve back-annotation results.

```
ERROR 251: The pre–route XNF file is required but
is not available.
```

The -a option was not used, or no file name was specified for the -a option. Check your command line and try again.

```
ERROR 252: The post–route XNF file is required
but is not available.
```

The -b option was not used, or no file name was specified for the -b option. Check your command line and try again.

```
ERROR 253: CLB/IOB detected in <post–route xnf
file>. Please run LCA2XNF -g and try again.
```

You must specify the -g option with LCA2XNF to generate a post-route XNF file that can be used by XNFBA with the -b option. The -g option ensures that the CLB and IOB symbols are not present in the XNF file.

```
ERROR 254:   Invalid part type <parttype> found
in file <file name>.
```

Mentor Graphics Interface/ Tutorial Guide

Index

Index

A

ABEL-HDL, 3-9
ABL2XNF, 6-5
Add force, 9-2
Add property, 3-7
AKA file, 10-17, 10-28
Attributes. See Properties

B

Back-annotation, 8-1, 10-15
BASE property, 4-9
BIT file, 7-1, 7-4, 7-7
Bitstream, 11-101
BLKNM property, 4-3, 4-4
BUFG, 5-7
Bus rippers, 3-5
    RULE property, 3-6

C

Calc_da, 11-9
CAP property, 4-9
Carry-logic, 3-4
chref, 2-8
Class P symbols, 1-8
CLB, 6-5, 8-3, 10-3, 10-5, 10-14
CLB/IOB properties
    Also see Properties
    Modifying, 4-9
CLOCK_OPT property, 5-17, 5-21
CMOS property, 4-9
COMP property, 4-1, 4-2
CONFIG property, 4-10
Configuring your system, 2-1
Converting designs, 2-3

D

DDP, 2-2
DECODE property, 4-4
DEF=PLD, 5-14
default.dvpt, 6-7
Delay option, 8-5
Design architect, 1-1
    See also PLD_DA
    Tutorial
        .PRO files, 11-83
        Adding buses, 11-35
        Adding labels, 11-56
        Adding nets, 11-36
        Adding pins to symbol, 11-22
        Adding ports, 11-42
        Adding text, 11-25
        Assigning pin locations, 11-69
        Bus rippers, 11-45
        calc_3k, 11-9
        Checking design rules, 11-96
        Checking for errors, 11-109
        Command summary, 11-111
        Completing ALU schematic, 11-50
        Configuring system, 11-5
        Configuring XMake with XDM, 11-80
        Constraints file, 11-81
        Copying components, 11-33
        Copying files, 11-13
        Creating ANDBLK2 symbol, 11-21

Creating ORBLK2 symbol, 11-28
Creating schematics for symbols, 11-30
Demonstration board, 11-97
Design description, 11-19
Download cable, 11-97
Downloading bitstream, 11-101
Entering commands, 11-18
Examining routed design, 11-93
FAST pads, 11-71
FD4CE symbol, 11-54
Files used, 11-8
Finding block, 11-95
Guide file, 11-106
Highlighting net, 11-96
I/O Flip-Flops, 11-72
Incremental changes, 11-105
Installing, 11-7
Inverting output display signals, 11-67
Labeling buses, 11-44
Labeling nets/buses, 11-39
Men2XNF8, 11-83
Mentor graphics variables, 11-6
Net connections, 11-38
Opening calc schematic, 11-63
Optimizing for XC4000, 11-73
Placing library components, 11-53
Placing user-created components, 11-51
PLD_XMake, 11-85
RAM stack, 11-75
Required software, 11-5
Saving ALU schematic, 11-58
Saving calc schematic, 11-73
Saving symbol, 11-27
Schematic layout, 11-69
Solution files, 11-8
Stack implementation, 11-74
State machine, 11-77
Targeting for XC4000, 11-14
Testing design, 11-103
Translating design, 11-108
Using function keys, 11-17
Using mouse, 11-16
Viewing primitive, 11-60
Viewing RPM, 11-61
Viewing soft macro, 11-59
Wide-edge decoders, 11-78
XC3000 demo board, 11-101
XC3000 oscillator, 11-64
XC3000/XC4000 demo board, 11-99
XC4000 demo board, 11-100
XC4000 oscillator, 11-65
Xilinx library elements, 11-58
Design DataPort, 2-2
Design flow, 1-1, 11-2
Design manager, 1-1
Defining the interface, 1-3
Icons, 1-3
See also PLD_DMGR
Design rule check, 7-8, 11-96
Design Viewpoint Editor
See PLD_DVE
Design.out file, 7-2
Do file, 9-1
DOUBLE property, 4-4
DRC, 11-96

E

EDIF file, 10-7, 10-22
EDIF primitive library, 10-19

EDIF2XNF, 10-5, 10-7, 10-8, 10-11, 10-18
  Error messages, A-1
  Options, 10-19
  Signal naming conventions, 3-3
  Symbol naming conventions, 3-3
  Syntax, 10-19
  Variables, 10-19
Editor icon, 1-5
ENWRITE, 10-22
  EDIF2XNF, 10-19
  Functional simulation, 10-5, 10-7, 10-8, 10-11
  Options, 10-22
  REF property, 4-1
  Syntax, 10-22
  Variables, 10-22
EPLD, 5-1
  3-state multiplexing, 5-7
  BUFG, 5-7
  Clocks, 5-7
  Components, 5-1, 5-8
    Counters, 5-11
    PLD, 5-12
  FastCLK, 5-7
  FOE line, 5-3
  Functional simulation, 6-2, 9-5, 10-2
  Global control nets, 5-7
  Input/output buffer connections, 5-2
  Output buffers and 3-States, 5-3
  Primitives and macros, 5-13
  XC7000 library, 5-1
EQN, 6-5, 8-3, 10-5
EQUATE_F property, 4-10
EQUATE_G property, 4-10
Error Messages, A-1
EXT records, 10-20

F

F attribute, 5-22
FAST property, 4-8
FastCLK, 5-7, 5-21
FILE property, 4-2, 6-5, 10-3
  Functional simulation, 10-8, 10-11
  Timing simulation, 10-15
FITNET, 10-13
Fitter, 3-8
Flip–flops, 9-2
FNCSIM8, 6-4
  Manual translation, 10-1
  Options, 10-23
  See also Functional simulation
  See also PLD_FNCSIM8
  Syntax, 10-22
  Variables, 10-23
fncsim8.log, 6-6
fncsim8.sh, 6-6
FOE_OPT property, 5-16, 5-21
Functional simulation, 1-2, 6-1
  Auto generate, 6-5
  EPLD, 6-2
  Manual translation, 10-2
  ROMs, 9-3
  See also PLD_FNCSIM8, FNCSIM8
  Use original, 6-5

G

GCLK, 11-64
gen_lib, 2-2
Gen_Sch8
  Error messages, A-4
  Functional simulation, 6-5, 10-5, 10-7, 10-8, 10-11
  Options, 10-24
  Syntax, 10-23
  Timing simulation, 10-17, 10-18
  Variables, 10-23
Gen_Sym8, 3-9
  EPLD, 5-14
  Error messages, A-5
Globalresetb, 9-2, 9-5
Globalsetreset, 9-2, 9-5
Guide file, 7-3

H

H attribute, 5-23
HM2RPM, 3-4
HU_SET property, 4-8

I

IBUF, 5-2
Implementation, 1-2, 7-1
  Manual translation, 10-11
    EPLD, 10-13
    FPGA, 10-11
  See also PLD_XEMake
  See also PLD_XMake
Incremental design, 11-4
INIT property, 4-5, 9-2, 9-3
Input/Output properties. See Properties.
INST property, 4-2
Intel Hex file, 7-1, 7-7, 10-13
INTERNAL property, 4-8
IOB, 6-5, 8-3, 10-3, 10-5, 10-14
IOB flip–flops, 9-2
IPAD, 5-2

L

Latches, 9-2
LCA file, 7-1, 7-4, 7-7, 8-1, 8-3
  Timing simulation, 10-15
LCA2XNF, 6-5, 10-5, 10-15, 10-16
  Program description, 10-24
LD_LIBRARY_PATH, 2-2, 11-6, 12-3, 16-2
listref, 2-8
LOC (EPLD) property, 5-18
LOC property, 4-6
  Adding to symbols, 4-6
Location maps, 2-2
Location properties. See Properties.
Logic cell array, 7-1
LOGIC_OPT property, 5-16, 5-20
LOWPWR property, 5-19
LOWPWR=ALL, 5-16

M

Macros
  EPLD, 5-13
    User-defined, 5-14
  FPGA, 3-4
    Hard, 3-4
    HM2RPM, 3-4
    RPM, 3-4
    Soft, 3-4
MAK file, 7-3, 7-6, 7-8
MakeBits, 7-4
Manual translation, 10-1
  Program summary, 10-18
MAP2LCA, 10-15, 10-17
Map-FILE=-then-merge, 7-4
Map-then-merge, 7-4
MBA file, 10-15
MBAPP, 10-15
MEDFAST property, 4-9
MEDSLOW property, 4-9
MemGen, 3-9, 6-3, 8-3, 10-3, 10-14
Men2XNF8, 6-1
  Log file, 7-8
  Manual translation, 10-1
  Options, 10-25
  See also PLD_Men2XNF8
  Syntax, 10-24
  Variables, 10-24
men2xnf8.log, 6-7, 7-8
men2xnf8.sh, 6-7
Merge-then-map, 7-4
MGC_GENLIB, 2-2, 11-6, 12-3, 16-2
MGC_HOME, 2-1, 11-6, 12-3, 16-2
MGC_LOCATION_MAP, 2-2, 11-6, 12-3, 16-2
MGC_WD, 2-2, 12-4, 16-3
MINIMIZE property, 5-16, 5-20
Models
  QuickSim II simulation, 9-1
Modify property, 3-8

MRINPUT property, 5-16, 5-23

N

Naming conventions, 3-1
Net properties. See Properties.
NET property, 4-13
NETFLAG (EPLD) property, 5-22
NETFLAG property, 4-13
Nets
  Analyzing, 9-1
NODELAY property, 4-8

O

Obsolete libraries, 3-4
OBUF, 5-2
OPAD, 5-2
Open sheet command, 3-7
OPT property, 5-19
Out file, 7-8

P

PALASM, 5-17
Part type, 10-20
PINTYPE property, 4-2
PLD, 1-1, 3-7, 5-1
PLD attribute, 5-17
PLD equation file, 5-12
PLD Functional Simulation
  Dialog box, 6-4
PLD symbol, 3-10
PLD Timing Simulation
  Dialog box, 8-2
PLD XNF Translation
  Dialog box, 6-2
PLD=, 3-10
PLD_DA, 1-1, 1-6
  Entering, 2-5
  See also Design architect
PLD_DMGR, 1-4
  Running applications, 2-4
  See also Design manager
  Starting, 2-3
PLD_DVE
  Design manager, 1-6
  Dialog box, 6-9
  Error messages, A-7
  Functional simulation, 10-5, 10-7, 10-8, 10-11
  Options, 10-25
  PINTYPE property, 4-2
  Syntax, 10-25
  Variables, 10-25
PLD_DVE_BA, 10-15
  Error messages, A-6
  Options, 10-26
  Syntax, 10-26
  Variables, 10-26
PLD_DVE_SIM, 10-3
  Error messages, A-7
  Functional simulation, 10-5, 10-7, 10-9, 10-11
  Options, 10-27
  Syntax, 10-26
  Timing simulation, 10-15, 10-17, 10-18
  Variables, 10-26
PLD_FNCSIM8, 1-2, 1-6, 6-1, 6-4, 6-6
  See also FNCSIM8
  See also Functional simulation
PLD_Men2XNF8, 1-6, 6-1, 7-1, 11-83
  Output files, 11-84
  See also Men2XNF8
PLD_TIMSIM8, 1-2, 1-7, 8-1
  See also Timing simulation, TIMSIM8
PLD_XDM, 1-7
PLD_XEMake, 1-2, 1-7, 7-1, 7-5
  Dialog box, 7-5
  MAK file, 7-6
  Output files, 7-7
  See also XEMake
  Target file, 7-6
PLD_XMake, 1-2, 1-7, 7-1, 11-85
  Dialog box, 7-1
  Output files, 7-7, 11-87
  See also XMake

Target file, 7-4
PLUSASM, 3-10, 5-1, 5-12, 10-13
PLD attribute, 5-17
User-defined primitives, 5-14
PPR, 1-8, 9-3
Preload
Initialization, 9-3
PRELOAD_OPT, 9-4
PRELOAD_OPT property, 5-17, 5-21
PRG file, 7-7, 10-13
Primitives
EPLD, 5-13
User-defined, 5-14
FPGA, 3-4
PRLD
Equations, 5-22
Forcing, 9-5
Initialization, 9-3
Programmable logic device, 1-1
Prolink, 1-2
Properties, 3-7, 4-1
Adding, 3-7
Also see specific property.
EPLD, 5-15
Adding, 5-16
CLOCK_OPT, 5-21
FOE_OPT, 5-21
Global, 5-16
LOC=pin_name, 5-18
LOGIC_OPT, 5-20
LOWPWR, 5-19
MINIMIZE, 5-20
MRINPUT, 5-23
NETFLAG, 5-22
OPT, 5-19
PLD=file_name, 5-17
PRELOAD_OPT, 5-21
REG_OPT, 5-21
UIM_OPT, 5-20
FPGA, 4-1
BASE, 4-9
BLKNM, 4-3, 4-4
CAP, 4-9
CMOS, 4-9
COMP, 4-1, 4-2
CONFIG, 4-10
DECODE, 4-4
DOUBLE, 4-4
EQUATE_F, 4-10
EQUATE_G, 4-10
FAST, 4-8
FILE, 4-2
HU_SET, 4-8
INIT, 4-5
INST, 4-2
INTERNAL, 4-8
LOC, 4-6
MEDFAST, 4-9
MEDSLOW, 4-9
NET, 4-13
NETFLAG, 4-13
NODELAY, 4-8
PINTYPE, 4-2
REF, 4-1
RES, 4-9
RLOC, 4-7
RLOC_ORIGIN, 4-8
RLOC_RANGE, 4-8
SLOW, 4-8
TNM, 4-11
TSidentifier, 4-11
TTL, 4-9
U_SET, 4-7

USE_RLOC, 4-7
Modifying, 3-8
PRP file, 7-8

Q

QuickPart tables, 9-1
QuickPath, 1-5, 8-6
Dialog box, 8-6
QuickSim II, 6-6, 9-1
Analyzing nets, 9-1
Dialog box, 6-7, 8-4
Functional simulation, 6-1, 6-7
Icon, 1-5
Options, 10-27
Simulation models, 9-1
Syntax, 10-27
Timing simulation, 8-1, 8-3
Tutorial, 12-1
Adding traces manually, 12-12
Asserting global reset, 12-15
Asserting global set reset, 12-16
Assigning values to clock, 12-13
Command file, 12-28
Command summary, 12-29
Design description, 12-17
Dofile, 12-29
FNCSIM8, 12-29
Functional simulation, 12-6
Installing, 12-4
Men2XNF8, 12-29
Opening list window, 12-10
Opening trace window, 12-10
PLD_FNCSIM8, 12-7
PLD_Men2XNF8, 12-7
PLD_TIMSIM8, 12-27
Required software, 12-2
Saving waveform data, 12-24
Selecting nets for simulation, 12-8
Simulating the circuit, 12-18
Timing simulation, 12-27
Timsim8.log file, 12-28
Using the transcript, 12-26
Viewing calc schematic, 12-7
Variables, 10-27

R

RAMs, 4-5
REF property, 4-1
REG_OPT property, 5-17, 5-21
Relationally placed macro
See RPM
Report file, 7-8
RES property, 4-9
Retargeting design, 2-6
Rip component, 3-5
RLOC property, 4-7
RLOC_ORIGIN property, 4-8
RLOC_RANGE property, 4-8
ROM
INIT property, 4-5
Simulation, 9-3
RPM, 1-8, 3-4
RPT file, 7-8

S

Schematic
Creating, 3-6
Signature, 7-7
SIMDIR, 10-7, 10-11, 10-29, 13-10
Simulation
ROMs, 9-3
Simulation models, 9-1
SLOW property, 4-8
Symbol properties. See Properties.
Symbols
Adding pins, 11-22
Syntax conventions, 10-2
System V Bourne Shell, 10-1

T

TIMESPEC primitives, 4-12
Timing simulation, 1-2, 8-1
- Auto generate, 8-3
- Manual translation, 10-14
- Output files, 8-4
- See also PLD_TIMSIM8, TIMSIM8
- Use original, 8-3

Timing specification properties
- Adding TIMESPEC / TIMEGRP, 4-11
- Adding TS flag, 4-12
- See also Properties
- TS Flag, 4-11

TIMSIM8, 8-1
- Manual translation, 10-1
- Options, 10-28
- Syntax, 10-27
- Variables, 10-28

timsim8.log, 8-4
timsim8.sh, 8-4
TNM property, 4-11
Tri-state (3-state), 9-3
TS flag, 4-11, 4-12
TSidentifier property, 4-11
TTL property, 4-9

U

U_SET property, 4-7
UIM_OPT property, 5-16, 5-20
UNAKAXNF, 10-15, 10-17
- Error messages, A-7
- Options, 10-28
- Syntax, 10-28

Unified libraries, 1-8, 3-4
- Retargeting design, 2-6

USE_RLOC property, 4-7

V

VCC symbol
- EPLD, 5-15

Viewpoint, 6-5
- Back-annotation, 6-9, 10-26
- Default, 10-26
- Functional simulation, 6-4
- Simulation, 6-9, 10-7
- Timing simulation, 8-3
- XNF, 6-9, 10-25

Visible option, 8-5
VMD file, 7-7
VMH file, 7-6, 7-7
VMH / VMD file, 5-14, 8-1, 8-3
VMH / VMD files, 3-9, 7-1
VMH2XNF, 10-18
- Program description, 10-29

X

XACT Design Editor, 11-93, 15-29
XACT libraries, 3-4, 3-5
XACT-Performance, 15-1
- Tutorial, 15-1
  - Adding TNM, 15-10
  - ALUFF class, 15-15
  - Analyze mode, 15-31
  - Analyze mode output, 15-31
  - Analyzing design, 15-29
  - Clock speed, 15-11
  - ClockToPad option, 15-28, 15-38
  - ClockToSetup option, 15-28, 15-38
  - CTLFF class, 15-16
  - Default timing, 15-9, 15-11
  - EditLCA, 15-41
  - EXCEPT statement, 15-6
  - FailedSpec option, 15-28, 15-32
  - Flagblk option, 15-27, 15-29
  - Floorplanning, 15-9
  - FromFF option, 15-28, 15-40
  - Grouping symbols, 15-3
  - IGNORE value, 15-9
  - Implementation, 15-22
  - INFFS class, 15-14

Installing, 15-2
Maxpaths, 15-39
PadToPad, 15-38
PadToPad option, 15-28
PadToSetup option, 15-28, 15-38
PLD_Men2XNF8, 15-23
PLD_XMake, 15-23
PLD_XMake output, 15-23
PPR, 15-1, 15-8, 15-22
PPR log file, 15-24
Predefined class, 15-5
Query template, 15-30
Required software, 15-2
Running XDelay, 15-29
SD/RD pins, 15-30
SelectSpec option, 15-28, 15-32
Speed grade, 15-9
Stacker class, 15-14
STFF class, 15-16
TIMEGRP, 15-3, 15-5, 15-17
TIMESPEC, 15-3, 15-7, 15-11
TIMESPEC constraints, 15-19
TimeSpec mode, 15-32
TimeSpec output, 15-33
Timing constraints, 15-8, 15-13
Timing specifications, 15-7
TNM attribute, 15-3
TNM groups, 15-14
ToFF option, 15-28, 15-40
TSMaxpaths option, 15-33
Using XACT-Performance, 15-8
XDelay, 15-1, 15-27
XDelay mode, 15-38
XNFPrep, 15-24
XRP file, 15-37

XAS file, 3-9
X-BLOX, 1-8, 3-4, 10-3, 10-7
  Functional simulation, 6-6
  Libraries, 3-4
  Optimization, 7-3
  Program description, 10-29
  Syntax, 10-29
  Timing simulation, 8-3
  Tutorial, 13-1
    ADD_SUB, 13-9
    Adding X-BLOX module, 13-3
    ANDBUS, 13-9
    ASYNC_VAL, 13-8
    Back-annotation, 13-21
    BLOXSOLN, 13-11, 13-13
    BOUNDS attribute, 13-6
    Bus definition, 13-7
    Bus widths, 13-6
    BUS_DEF, 13-6
    BUS_IF symbol, 13-6
    Buses, 13-5
    Completing ALU_BLOX, 13-5
    DATA_REG, 13-8
    Design description, 13-2
    Dofile, 13-17, 13-24
    Download to demo board, 13-20
    ENCODING attribute, 13-6
    FNCSIM8 output, 13-13
    Functional simulation, 13-10
    Gen_Sch8, 13-13
    Implementation, 13-18
    INVMASK, 13-9
    Library, 13-1, 13-8
    Men2xnf8.log file, 13-11
    MUXBUSx, 13-9
    ORBUS, 13-9

PLD_FNCSIM8, 13-9, 13-10, 13-13
PLD_Men2XNF8, 13-11
PLD_TIMSIM8, 13-21
PLD_TIMSIM8 output, 13-21
PLD_XMake, 13-9, 13-18
PLD_XMake output, 13-18
QuickSim II, 13-17, 13-23
Required software, 13-1
SIMDIR, 13-10
Symbols, 13-8
SYNC_VAL, 13-8
Timing simulation, 13-21
TIMSIM8, 13-21
Viewing ALU_BLOX, 13-3
Viewing schematic, 13-15
XBLXGS, 13-10, 13-13
Variables, 10-29
X-BLOX library, 13-1, 13-8
XBLXGS, 10-7, 10-11
Error messages, A-9
Options, 10-31
Syntax, 10-30
Variables, 10-30
XC2000/XC3000 designs
Globalresetb, 9-2, 9-5
XC4000 designs
Globalsetreset, 9-2, 9-5
XC7000 library, 3-10, 5-1
XDE, 11-93
Checking design rules, 11-96
Finding block, 11-95
Highlighting net, 11-96
Invoking, 11-93
XDelay, 10-15, 10-16, 15-1, 15-27
Program description, 10-31
XDM, 1-7
XEPLD
Tutorial, 16-1
.err file, 16-38
.lga file, 16-38
.lgc file, 16-38
.map file, 16-38
.par file, 16-38
.pin file, 16-38
.res file, 16-38
Add bus, 16-41
Add list, 16-41
Add pins, 16-46
Add symbol name, 16-49
Add trace, 16-41
Adding bus names, 16-25
Adding names, 16-22
Adding symbols, 16-16
Assembler log report, 16-38
Assigning signals, 16-31
Attributes, 16-28
Boolean equations, 16-32, 16-33
Bus command, 16-18
Bus names, 16-25
Bus rippers, 16-21, 16-26
Check symbol, 16-49
Checking design, 16-31
Complete UART schematic, 16-30
Creating a bus, 16-18
Creating a schematic, 16-15
Creating wires, 16-20
Custom symbols, 16-45
Delete force, 16-41
Deleting symbols, 16-17
Demonstration summary, 16-7
Design architect, 16-12
Design description, 16-5
Do file, 16-40

Draw symbol, 16-46
Drawing design, 16-12
EQUATIONS keyword, 16-33
Example files, 16-6
Fitter, 16-22, 16-37
Fitter error report, 16-38
Fitting the design, 16-37
Force command, 16-41
Global property, 16-28
Installing, 16-1
Intel Hex file, 16-37
JEDEC, 16-37
Lengthen pins, 16-48
List file, 16-43
Logic optimizer report, 16-38
Mapping report, 16-38
Open symbol, 16-46
PALASM, 16-37
Partitioner report, 16-38
Pinlist report, 16-38
Placing symbol, 16-50
PLD equations, 16-32
PLD file, 16-32
PLD=filename, 16-28
PLD_DMGR, 16-11
PLD_FNCSIM8, 16-54
PLD_Men2XNF8, 16-37, 16-54
PLD_TIMSIM8, 16-42
PLD_XEMake, 16-37
PLD_XEMake reports, 16-38
PLUSASM, 16-32
Preparing system, 16-10
Properties, 16-28
QuickSim II, 16-40, 16-54
RCVR.PLD file, 16-32
RCVRSUB, 16-51
Required software, 16-1
Resource report, 16-38
Run command, 16-42
Save symbol, 16-49
Saving design, 16-32
Session 1, 16-10
Session 2, 16-12
Session 3, 16-32
Session 4, 16-37
Session 6, 16-45
Session overview, 16-9
Simulation, 16-39, 16-54
Symbol window, 16-45
Trace file, 16-44
UART, 16-5
UART schematic, 16-13
Unified library, 16-16
Viewing design, 16-13
Wire command, 16-20
XACT library, 16-16
Zoom level, 16-15
XFF file, 10-7
XGS file, 10-7, 10-29
Xilinx ABEL, 3-9, 6-5, 10-3, 10-14
Adding symbol to schematic, 3-9
Creating a symbol, 3-9
Timing simulation, 8-3
Tutorial, 14-1
ABL2XNF, 14-12
Adding STAT_ABL to calc, 14-13
Back-annotation, 14-23
Creating a symbol, 14-12
Design description, 14-3
Dofile, 14-20, 14-25
File property, 14-14

FNCSIM8, 14-15
Functional simulation, 14-15
Gen_Sch8, 14-15
Gen_Sym8, 14-12
Implementation, 14-20
Installing, 14-2
PLD_FNCSIM8, 14-15, 14-17
PLD_FNCSIM8 output, 14-18
PLD_Men2XNF8, 14-16
PLD_Men2XNF8 output, 14-16
PLD_TIMSIM8, 14-23
PLD_TIMSIM8 output, 14-24
PLD_XMake, 14-20
PLD_XMake output, 14-22
QuickSim II, 14-20, 14-25
Required software, 14-1
STAT_ABL, 14-3, 14-14
Synthesizing Xilinx ABEL, 14-11
Timing simulation, 14-23
Verifying design with demo board, 14-23
Verifying STAT_ABL, 14-11
Viewing schematic, 14-18
Viewing STAT_ABL.ABL, 14-3
XAS file, 14-12
Xilinx ABEL output, 14-7
XSF file, 14-12
Xilinx attributes. See Properties
Xilinx design manager. See XDM, PLD_XDM
Xilinx libraries, 3-4
XMake
MAK file, 7-3
XNF file, 6-1, 6-5, 7-1, 10-7
XNFBA, 10-15, 10-17
Error messages, A-10
Syntax, 10-31
Variables, 10-31
XNFMerge, 10-5, 10-7, 10-11, 10-15
Program description, 10-34
Tutorial design files, 11-10
XNFPrep, 7-8, 10-7, 10-11, 10-29
Program description, 10-35
XSF file, 3-9
EPLD, 5-14
XTF file, 10-7, 10-29