

Mentor Graphics Interface/ Tutorial Guide

Introduction

Getting Started

Schematic Designs

HDL Designs

***Mixed Designs with VHDL
on Top***

***Mixed Designs with
Schematic on Top***

Advanced Techniques

Manual Translation

Schematic Design Tutorial

***Schematic-on-Top with
VHDL Tutorial***



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

XILINX, XACT, XC2064, XC3090, XC4005, XC5210, XC-DS501, FPGA Architect, FPGA Foundry, NeoCAD, NeoCAD EPIC, NeoCAD PRISM, NeoROUTE, Timing Wizard, and TRACE are registered trademarks of Xilinx, Inc.



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

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

All other trademarks are the property of their respective owners.

Xilinx, Inc. does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx, Inc. reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx, Inc. will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx, Inc. devices and products are protected under one or more of the following U.S. Patents: 4,642,487; 4,695,740; 4,706,216; 4,713,557; 4,746,822; 4,750,155; 4,758,985; 4,820,937; 4,821,233; 4,835,418; 4,855,619; 4,855,669; 4,902,910; 4,940,909; 4,967,107; 5,012,135; 5,023,606; 5,028,821; 5,047,710; 5,068,603; 5,140,193; 5,148,390; 5,155,432; 5,166,858; 5,224,056; 5,243,238; 5,245,277; 5,267,187; 5,291,079; 5,295,090; 5,302,866; 5,319,252; 5,319,254; 5,321,704; 5,329,174; 5,329,181; 5,331,220; 5,331,226; 5,332,929; 5,337,255; 5,343,406; 5,349,248; 5,349,249; 5,349,250; 5,349,691; 5,357,153; 5,360,747; 5,361,229; 5,362,999; 5,365,125; 5,367,207; 5,386,154; 5,394,104; 5,399,924; 5,399,925; 5,410,189; 5,410,194; 5,414,377; 5,422,833; 5,426,378; 5,426,379; 5,430,687; 5,432,719; 5,448,181; 5,448,493; 5,450,021; 5,450,022; 5,453,706; 5,466,117; 5,469,003; 5,475,253; 5,477,414; 5,481,206; 5,483,478; 5,486,707; 5,486,776; 5,488,316; 5,489,858; 5,489,866; 5,491,353; 5,495,196; 5,498,979; 5,498,989; 5,499,192; 5,500,608; 5,500,609; 5,502,000; 5,502,440; 5,504,439; 5,506,518; 5,506,523; 5,506,878; 5,513,124; 5,517,135; 5,521,835; 5,521,837; 5,523,963; 5,523,971; 5,524,097; 5,526,322; 5,528,169; 5,528,176; 5,530,378; 5,530,384; 5,546,018; 5,550,839; 5,550,843; 5,552,722; 5,553,001; 5,559,751; 5,561,367; 5,561,629; 5,561,631; 5,563,527; 5,563,528; 5,563,529; 5,563,827; 5,565,792; 5,566,123; 5,570,051; 5,574,634; 5,574,655; 5,578,946; 5,581,198; 5,581,199; 5,581,738; 5,583,450; 5,583,452; 5,592,105; 5,594,367; 5,598,424; 5,600,263; 5,600,264; 5,600,271; 5,600,597; 5,608,342; 5,610,536; 5,610,790; 5,610,829; 5,612,633; 5,617,021; 5,617,041; 5,617,327; 5,617,573; 5,623,387; 5,627,480; 5,629,637; 5,629,886; 5,631,577; 5,631,583; 5,635,851; 5,636,368; 5,640,106; 5,642,058; 5,646,545; 5,646,547; 5,646,564; 5,646,903; 5,648,732; 5,648,913; 5,650,672; 5,650,946; 5,652,904; 5,654,631; 5,656,950; 5,657,290; 5,659,484; 5,661,660; 5,661,685; 5,670,897; 5,670,896; RE 34,363, RE 34,444, and RE 34,808. Other U.S. and foreign patents pending. Xilinx, Inc. does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. Xilinx, Inc. assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx, Inc. will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

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

Copyright 1991-1997 Xilinx, Inc. All Rights Reserved.

Preface

About This Manual

This manual explains how to use Version M1 of the Xilinx/Mentor Graphics Interface software with Mentor Graphics® software versions B.1, B.2, B.3, and B.4.

Manual Contents

This manual covers the following topics.

- Chapter 1, “**Introduction**,” describes the Mentor Graphics Design Manager™ Interface, the Xilinx design flow, key features, inputs and outputs, and the architectures with which they work.
- Chapter 2, “**Getting Started**,” describes how to configure your system for the Mentor Graphics Design Manager, and how to invoke and exit the Mentor Graphics Design Manager.
- Chapter 3, “**Schematic Designs**,” describes how to use the Mentor Graphics Design Manager and Design Architect™ to design with pure schematic designs. It covers, schematic design entry, functional simulation, implementation, and timing simulation.
- Chapter 4, “**HDL Designs**,” describes how to use the Mentor Graphics Interface to design with pure HDL designs. It covers, HDL design entry, functional simulation, implementation, and timing simulation.
- Chapter 5, “**Mixed Designs with VHDL on Top**,” describes how to use the Mentor Graphics Interface to design with mixed schematic and VHDL designs with VHDL on Top. It covers, design entry, functional simulation, implementation, and timing simulation.

- Chapter 6, “**Mixed Designs with Schematic on Top,**” describes how to use the Mentor Graphics Interface to design with mixed schematic and VHDL designs with schematic on top. It covers, design entry, functional simulation, implementation, and timing simulation.
- Chapter 7, “**Advanced Techniques,**” describes useful design and simulation techniques that were not covered in the other sections of this manual.
- Chapter 8, “**Manual Translation,**” describes how to manually process your design from the operating system command line.
- Chapter 9, “**Schematic Design Tutorial,**” steps you through a typical FPGA or CPLD design procedure from schematic entry to completing a functioning device using the Mentor Graphics Design Architect configured for Xilinx designs. It also steps you through both a functional and a timing simulation of an FPGA or CPLD design using the Mentor Graphics QuickSim II™.
- Chapter 10, “**Schematic-on-Top with VHDL Tutorial,**” guides you through a typical FPGA and CPLD design procedure from schematic entry with instantiated HDL to completion of a functioning device using the Mentor Graphics Design Architect configured for Xilinx designs.

Conventions

Typographical

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

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

```
speed grade: -100
```

- **Courier bold** indicates literal commands that you enter in a syntactical statement.

```
rpt_del_net=
```

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

File → **Open**

- *Italic font* denotes the following items.
 - Variables in a syntax statement for which you must supply values
`edif2ngd design_name`
 - References to other manuals
See the *Development System Reference Guide* for more information.
 - Emphasis in text

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

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

`edif2ngd [option_name] design_name`

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

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

`lowpwr = {on|off}`

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

`symbol editor_name [bus|pins]`

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

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

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

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

Online Document

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

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

Figure 1-1 Naming Conventions



Iconized tables are identified by the table icon.

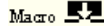
Table 13-14 Carry Modes



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



The DynaText footnote icon displays next to the footnoted text.



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

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

Contents

Preface

About This Manual	iii
Manual Contents	iii

Conventions

Typographical.....	v
Online Document	vi

Chapter 1 Introduction

Architecture Support	1-1
Platform Support	1-2
Library Support.....	1-2
Features	1-2
Mentor Software Release Support.....	1-2
Added HDL Support.....	1-2
QuickHDL and QuickHDL PRO	1-3
VHDL Gate-Level Simulation Support	1-3
Verilog Gate-Level Simulation Support.....	1-3
Links to the Xilinx Synopsys Interface (XSI)	1-3
Mentor Design Manager	1-4
Pld_da.....	1-5
Pld_dve.....	1-6
Pld_quicksim.....	1-6
Editor	1-6
QuickPath	1-6
LogiBLOX GUI	1-7
Gen_Arch.....	1-7
SysArch	1-7
Pld_edif2sim	1-7
Pld_edif2tim	1-7
Pld_xnf2sim	1-7

Pld_men2edif.....	1-8
QuickHDL	1-8
QuickHDL PRO.....	1-8
Pld_dsgnmgr.....	1-8
Pld_sg.....	1-8
New Models for LogiBLOX Modules	1-8
EDIF	1-9
Cross-Probing	1-9
Timing Simulation	1-9
Schematic Generator	1-9
Timing Constraints	1-10
Design Flows.....	1-10
Schematic Entry Design Flows	1-10
HDL Entry	1-14
Mixed Schematic and VHDL Flow with VHDL on Top	1-15
Mixed Schematic and VHDL Flow with Schematic on Top	1-16
Inputs	1-17
EDIF	1-17
XNF.....	1-17
Outputs.....	1-17
Files.....	1-18
Tutorials	1-19
Online Help	1-19

Chapter 2 Getting Started

Configuring Your System	2-1
Modifying Mentor Graphics Variables	2-2
Invoking the Design Manager	2-4
Invoking Applications in the Design Manager	2-4
Tools Window Icons.....	2-4
Navigator Window.....	2-4
Exiting the Design Manager	2-4

Chapter 3 Schematic Designs

Design Flows.....	3-1
Design Entry.....	3-1
Invoking Design Architect	3-1
Exiting Design Architect	3-3
Loading a Schematic	3-4
Creating the Design Component.....	3-5
Adding Components	3-5

Adding Xilinx library Components	3-5
Xilinx Libraries	3-5
Adding Properties	3-9
Properties	3-9
Adding Properties	3-10
Adding the Net Property to Nets	3-13
Modifying Property Values	3-13
Entering Timing Specifications	3-15
Creating New Groups from Existing Groups	3-16
Functional Simulation	3-16
Simulating Pure Schematic Designs	3-17
Creating the Viewpoint	3-17
Simulating the Design	3-19
Simulating Schematic Designs with LogiBLOX Elements	3-21
Simulating Schematic Designs with XNF Elements	3-21
Creating the Design Component	3-22
Converting the XNF File	3-22
Creating the Viewpoint	3-24
Simulating the Design	3-24
Simulating Schematic Designs with EDIF Elements	3-24
Creating the Design Component	3-25
Converting the EDIF File	3-25
Simulating the Design	3-27
Implementing Schematic Designs	3-27
Converting the EDDM Design	3-27
Implementing the Design	3-29
Timing Simulation for Schematic Designs	3-35
Creating the EDDM Model and the Viewpoint	3-36
Simulating the Design	3-38
Cross-Probing	3-40
Performing a Timing Analysis	3-42

Chapter 4 HDL Designs

The Design Flow	4-1
HDL Design Entry	4-3
Overview of HDL Design Entry	4-3
HDL Design Entry Stages	4-4
Stage 1: RTL Behavioral Code Development	4-5
Stage 2: Synthesis	4-6
LogiBLOX Design Entry	4-7
Unified Library Instantiated Components	4-8
Functional Simulation	4-8

Pre-Synthesis Functional Simulation	4-9
Post-Synthesis Functional Simulation.....	4-11
Optional Post Synthesis Functional Simulation	4-13
Design Implementation	4-15
Timing Simulation.....	4-25
Compiling the SIMPRIM Libraries.....	4-25
Passing Timing Generics to Special Cells—ROC, OSC, OSC4, and OSC5	4-25
Compiling the Design.....	4-27
Simulating the Design	4-28

Chapter 5 Mixed Designs with VHDL on Top

The Design Flow	5-1
Design Entry.....	5-2
Functional Simulation.....	5-8
Compiling the Design.....	5-8
Simulating the Design	5-8
Optional Post-Synthesis Functional Simulation	5-10
Design Implementation	5-11
Timing Simulation.....	5-20
Compiling the SIMPRIM Libraries.....	5-20
Passing Timing Generics to Special Cells—ROC, OSC, OSC4, and OSC5	5-20
Compiling the Design.....	5-22
Simulating the Design	5-23

Chapter 6 Mixed Designs with Schematic on Top

The Flow	6-1
Design Entry.....	6-2
VHDL Module Design Entry	6-3
Schematic Entry.....	6-4
Functional Simulation.....	6-5
Functional Simulation Before Synthesis	6-5
Functional Simulation After Synthesis	6-7
Design Implementation	6-9
Converting the EDDM Design.....	6-9
Implementing the Design	6-11
Timing Simulation.....	6-11

Chapter 7 Advanced Techniques

Retargeting the Design to a Different Family	7-1
--	-----

Merging Design Files from Other Sources	7-4
Simulation Models	7-4
Analyzing Nets from the Schematic	7-4
Setting Global Reset and 3-State Signals.....	7-5
FPGA Designs	7-5
CPLD Designs	7-6

Chapter 8 Manual Translation

Functional Simulation.....	8-1
Pure Schematic Designs.....	8-1
Schematic Designs with XNF Elements.....	8-1
Schematic Designs with LogiBLOX Elements	8-2
Mixed Schematic and VHDL with Schematic-on-Top Designs	8-2
Before Synthesis.....	8-2
After Synthesis.....	8-4
HDL-at-Top Designs	8-4
Pure HDL Designs	8-5
Design Implementation	8-5
Schematic Designs (FPGA).....	8-5
Schematic Designs (CPLD)	8-6
HDL-at-Top Designs	8-7
Pure HDL Designs	8-8
Timing Simulation.....	8-9
Schematic Designs	8-9
Pure HDL Designs	8-10
EDIF Method.....	8-10
VHDL/Verilog Method	8-11
Program Summary	8-11
CPLD	8-11
Dsgnmgr	8-11
EDIF2NGD.....	8-12
Editor.....	8-12
Gen_Arch.....	8-12
MAP	8-12
NGDAnno.....	8-12
NGDBuild	8-13
NGD2EDIF	8-13
PAR.....	8-13
Pld_da.....	8-13
Pld_dve	8-13
Pld_edif2sim	8-14
Pld_edif2tim	8-15

Pld_men2edif	8-15
Pld_quicksim	8-16
Pld_xnf2sim	8-17
QuickHDL	8-18
QuickHDL PRO	8-18
QuickPath	8-19
Qvhcom	8-19
Qvlcom	8-19
SysArch	8-19
Pld_sg	8-19

Chapter 9 Schematic Design Tutorial

Introduction	9-2
Required Background Knowledge	9-3
Design Flow	9-3
Software Installation	9-4
Required Software	9-4
Before Beginning the Tutorial	9-4
Installing the Tutorial	9-6
Standard Directory Structure	9-6
Tutorial Directory and Files	9-6
Starting the Design Manager	9-9
Tools Window	9-10
Navigator Window	9-10
Command Palette	9-10
Copying the Tutorial Files	9-11
Starting Design Architect	9-13
Using the Mouse in Design Architect	9-15
Left Mouse Button	9-15
Middle Mouse Button (Strokes)	9-15
Right Mouse Button	9-15
Using the Function Keys	9-16
Selecting Commands from the Menu Bar	9-16
Selecting Commands from the Palette	9-16
Entering Commands from the Keyboard	9-17
Cancelling Commands	9-17
Repeating Menu Commands	9-17
Manipulating the Screen	9-17
Targeting the Design for the XC9000 Family	9-17
Completing the Calc Design	9-20
Design Description	9-20
Creating the ANDBLK2 Symbol	9-22

Opening a Symbol Window	9-22
Creating the Symbol Outline	9-22
Adding Pins to the ANDBLK2 Symbol	9-22
Adding Text.....	9-26
Modifying Text Size	9-27
Saving the ANDBLK2 Symbol	9-28
Creating the ORBLK2 Symbol	9-28
Creating Schematics for ANDBLK2 Symbol	9-30
Opening a Schematic Window.....	9-30
Adding the First Component to a Schematic	9-30
Placing Additional Components.....	9-32
Copying a Component	9-33
Moving a Component.....	9-34
Adding Buses to a Schematic.....	9-35
Adding Nets to a Schematic	9-36
Completing the Net Connections	9-39
Increasing Text Size	9-40
Adding Ports	9-42
Labeling Ports.....	9-43
Saving the Schematic.....	9-44
Creating Schematics for ORBLK2 Symbol	9-45
Editing the ALU Schematic	9-47
Placing User-Created Components	9-49
Placing Library Components.....	9-51
Adding Nets, Buses, Ports and Labels	9-52
FD4CE and AND5B2	9-52
ANDBLK2 and ORBLK2	9-53
Adding Labels to Components.....	9-54
Saving the ALU Schematic	9-56
Exploring Xilinx Library Elements	9-56
Viewing a Xilinx Soft Macro Schematic.....	9-57
Viewing a Xilinx RPM (XC4000-Based Families Only)	9-57
Opening the Calc Schematic	9-61
Using the XC4000E Oscillator	9-61
Controlling FPGA/CPLD Layout from the Schematic.....	9-62
Assigning Pin Locations.....	9-62
Designating FAST Pads.....	9-64
Using the I/O Flip-Flops	9-65
Saving the Calc Schematic.....	9-66
Modifying the Design for Non-XC4000E/EX Devices.....	9-66
RAM Stack Implementation	9-66
Using the Device-Independent Register File	9-68

Removing the XC4000E Oscillator	9-69
Using LogiBLOX.....	9-71
Creating and Instantiating a LogiBLOX Module	9-71
Other Special Components	9-74
The STARTUP Block (Optional: XC4000E/EX and XC5200 only)	9-74
Adding the CONFIG Symbol (Optional)	9-76
Using a Constraints File	9-77
Performing Functional Simulation	9-78
Using Pld_dve	9-79
Invoking Pld_quicksim	9-80
Viewing the Calc Schematic	9-81
Selecting Nets for Simulation	9-82
Opening Trace and List Windows	9-84
Adding Traces Manually	9-85
Assigning Values to the Clock	9-87
Asserting Global Set/Reset (without STARTUP)	9-89
Asserting Global Set/Reset (with STARTUP)	9-90
Design Description	9-92
Simulating the Circuit	9-92
Saving the Results	9-98
Using the Transcript.....	9-100
Using Pld_men2edif	9-100
Examining Pld_men2edif Output Files	9-102
Using the Xilinx Design Manager	9-103
Performing Timing Simulation	9-109
Using Pld_edif2tim to Prepare a Timing Simulation.....	9-109
Examining the Pld_edif2tim.log File	9-110
Using Pld_dve	9-111
Invoking QuickSim for Timing Simulation	9-112
Examining Routed Designs with EPIC	9-115
Verifying the Design Using a Demonstration Board	9-116
Creating and Downloading the Bitstream	9-116
Making Incremental Design Changes	9-117
Making an Incremental Schematic Change	9-117
Translating the Incremental Design	9-119
Verifying the Change in the Demonstration Board.....	9-121
Command Summaries	9-121
XC4000E Command Summaries	9-121
Functional Simulation	9-121
Basic Translation	9-122
Timing Simulation	9-122
Incremental Translation	9-122

XC9000 Command Summaries	9-122
Functional Simulation	9-122
Basic Translation	9-123
Timing Simulation	9-123
Incremental Translation	9-123
Further Reading	9-123

Chapter 10 Schematic-on-Top with VHDL Tutorial

Introduction	10-1
Required Background Knowledge.....	10-3
Design Flow	10-3
Software Installation.....	10-4
Required Software	10-4
Before Beginning the Tutorial	10-4
Installing the Tutorial.....	10-5
Standard Directory Structure	10-5
Tutorial Directory and Files.....	10-6
Starting the Design Manager	10-7
Copying the Tutorial Files	10-8
Starting Design Architect.....	10-9
Completing the Calc Design.....	10-10
Design Description.....	10-11
Adding the SEG7DEC Component.....	10-12
Compiling the VHDL Entity	10-12
Linking a VHDL Entity to the ALU Component	10-18
Compiling the VHDL Entity	10-18
Using a Constraints File	10-22
Performing Functional Simulation	10-24
Using Pld_dve	10-24
Invoking QuickHDL Pro.....	10-25
Viewing the Calc Schematic	10-28
Viewing and Navigating the VHDL Hierarchy	10-29
Completing the Functional Simulation	10-32
Using Pld_men2edif.....	10-34
Examining Pld_men2edif Output Files.....	10-36
Using the Xilinx Design Manager	10-36
Performing Timing Simulation	10-42
Using Pld_edif2tim to Prepare a Timing Simulation.....	10-42
Examining the Pld_edif2tim.log File.....	10-43
Using Pld_dve	10-44
Invoking QuickSim for Timing Simulation	10-45
Examining Routed Designs with EPIC	10-48

Verifying the Design Using a Demonstration Board.....	10-48
Creating and Downloading the Bitstream	10-48
Command Summaries	10-49
XC4000E Command Summaries.....	10-50
Functional Simulation	10-50
Basic Translation	10-50
Timing Simulation	10-50
Further Reading	10-50

Introduction

This chapter describes the Mentor Graphics® Design Manager™ interface, a Mentor Graphics tool enhanced by the addition of Xilinx features.

You can invoke all individual tools from the Xilinx-enhanced Design Manager or from the shell.

This chapter contains the following sections:

- “Architecture Support” section
- “Platform Support” section
- “Library Support” section
- “Features” section
- “Design Flows” section
- “Inputs” section
- “Outputs” section
- “Files” section
- “Tutorials” section
- “Online Help” section

Architecture Support

You can use the Mentor interface with the following Xilinx architectures:

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

- XC5200
- XC9500/F

Note: You cannot mix old XC4000EX library components with XC4000X library components. Use Convert Design to convert XC4000EX designs to XC4000X before instantiating new XC4000X library components.

Platform Support

The Mentor interface is supported on Sun SPARCstations using either the Sun operating system version 4.1.3 and 4.1.4 or the Solaris operating system versions 2.4 and 2.5. It is also supported on HP workstations using the HP-UX operating system versions 9.05 and 10.02.

Library Support

The following libraries are available in the Mentor interface:

- Unified Libraries, which contain the symbol models for schematic entry and simulation
- SIMPRIM library, which contains the symbol models for timing (EDDM) simulation
- VITAL VHDL SIMPRIM library for top-down timing simulation
- Verilog SIMPRIM library for top-down Verilog timing simulation

Features

The following sections describe the major features available in this release.

Mentor Software Release Support

This interface supports the Mentor B.1, B.2, B.3, and B.4 software releases.

Added HDL Support

This release offers a number of features that allow you to process a design through a VHDL or Verilog netlist.

QuickHDL and QuickHDL PRO

This release supports the QuickHDL™ simulator, which simulates behavioral VHDL, Verilog, VHDL-based, and Verilog-based gate-level designs composed of SIMPRIM elements. In addition, LogiBlox elements can be simulated at the behavioral level.

It also supports QuickHDL PRO™ for mixed mode simulations for schematic-based and VHDL-based designs. QuickHDL PRO can invoke QuickHDL to simulate VHDL-based elements, or `pld_quicksim` to simulate Unified Libraries elements.

VHDL Gate-Level Simulation Support

This release supports VHDL simulation, including IEEE-standard 1076.4 VHDL libraries of SIMPRIM models. Xilinx implementation tools output timing simulation VHDL netlists by using structural VHDL models of SIMPRIM VHDL models and an SDF file.

Verilog Gate-Level Simulation Support

This release supports Verilog simulation, including Verilog libraries for use with SIMPRIM models. Xilinx implementation tools output timing Verilog netlists by using structural Verilog models with SIMPRIM Verilog models and an SDF file.

Links to the Xilinx Synopsys Interface (XSI)

The Mentor interface can accept Synopsys synthesized netlists in the form of SEDIF or SXNF files. It can also accept XNF and EDIF files from other synthesizers that are compatible with the Xilinx core software. These files can be directly submitted to the Xilinx Design Manager for placement and routing of the design.

You can also simulate these EDIF or SXNF files by submitting them to the `pld_edif2sim` or `pld_xnf2sim` utility, which creates EDDM components for use with `pld_quicksim`.

In addition, after place and route, you can output VHDL and Verilog netlists, which can be submitted to QuickHDL for simulation with SDF files providing the back-annotation information.

Mentor Design Manager

The Mentor Graphics Design Manager is an 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 and a menu item.

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 Mentor Design Manager.

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

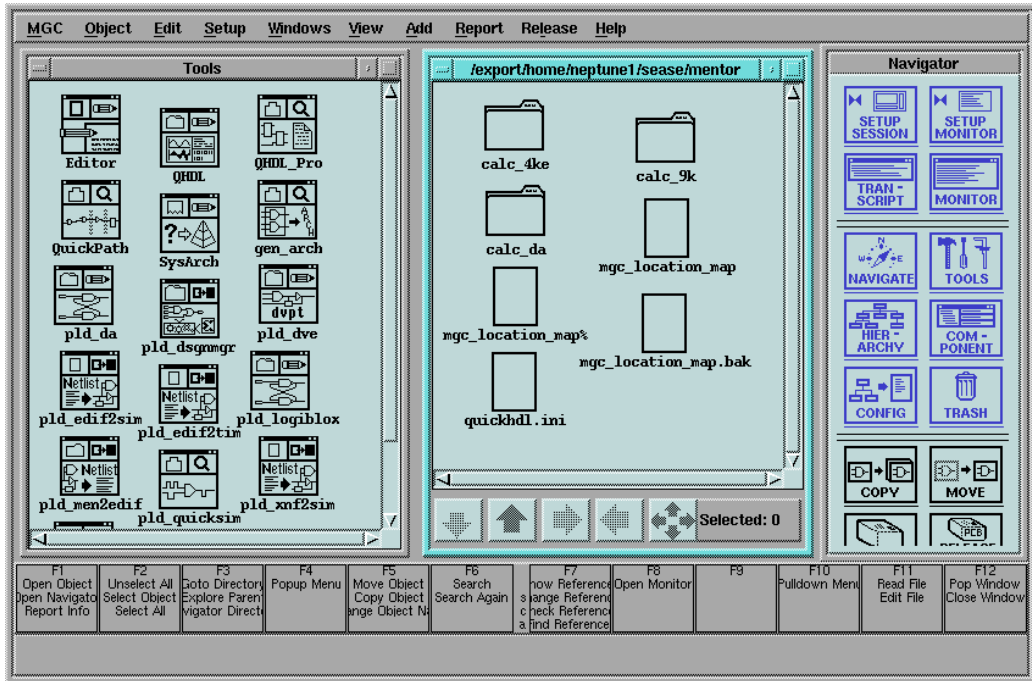


Figure 1-1 Mentor Design Manager Window

The Tools window contains icons representing all the Mentor Graphics and Xilinx applications that 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 Design Manager palette provides easy access to the most commonly used Design Manager menu items.

The remainder of this section briefly describes the icons in the Tools window and the Mentor programs they represent. The tools with names that begin with PLD are configured through scripts for working with Xilinx designs.

Pld_da

Pld_da is Mentor's Design Architect[®], a schematic editor configured for Xilinx designs. The Xilinx-configured Design Architect is identical to the Mentor Graphics version except for the addition of a Xilinx

library of primitives, macros, and utilities such as Convert Design. Refer to the “[Design Entry](#)” section of the “[Schematic Designs](#)” chapter in this manual and the “[Schematic Design Tutorial](#)” chapter 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

Pld_dve is the Mentor Graphics Design Viewpoint Editor (DVE) configured for Xilinx designs. When you invoke this application from within the Mentor Design Manager, a dialog box appears and you are asked to create either a simulation or custom viewpoint. Refer to the “[Functional Simulation](#)” section of the “[Schematic Designs](#)” chapter and the “[Timing Simulation for Schematic Designs](#)” section of the “[Schematic Designs](#)” 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_quicksim

Pld_quicksim is an interactive logic simulator that performs functional or timing simulation on your designs. For more information on pld_quicksim, refer to the “[Functional Simulation](#)” section of the “[Schematic Designs](#)” chapter, the “[Timing Simulation for Schematic Designs](#)” section of the “[Schematic Designs](#)” chapter, and the “[Schematic Design Tutorial](#)” chapter in this manual. For a detailed description of pld_quicksim, refer to the Mentor Graphics *QuickSim II User’s Manual*.

Editor

The Editor icon represents the Mentor Graphics Notepad editor. Notepad is a full-featured, window-based text editor. For more information on Notepad, refer to the Mentor Graphics *Notepad User’s and Reference Manual*.

QuickPath

QuickPath™ performs static and slack timing analysis on designs. For more information on QuickPath, refer to the “[Timing Simulation for Schematic Designs](#)” section of the “[Schematic Designs](#)” chapter

and TMthe “**Schematic Design Tutorial**” chapter in this manual. For a detailed description of QuickPath, refer to the Mentor Graphics *QuickPath User's and Reference Manual*.

LogiBLOX GUI

This is a stand-alone Xilinx tool for generating VHDL and Verilog models of LogiBlox components. Schematic models can be created by invoking LogiBLOX from within pld_da under the Xilinx Libraries Palette menu.

Gen_Arch

Gen_Arch creates a VHDL architecture from a Mentor schematic (EDDM) component for use in mixed schematic and HDL simulations within QuickHDL Pro.

SysArch

SysArch is the System ArchitectTM, which creates system-level designs and outputs synthesizable VHDL.

Pld_edif2sim

Pld_edif2sim is a utility that converts a Mentor, Synopsys, or other Xilinx compatible EDIF file into a Mentor EDDM single-object simulation model, VHDL netlist, or Verilog netlist. Pld_edif2sim is for functional simulation only.

Pld_edif2tim

Pld_edif2tim is the Mentor EDIF netlist reader, which converts a placed and routed EDIF netlist to a Mentor single-object EDDM file that can be submitted to pld_quicksim for timing simulation.

Pld_xnf2sim

Pld_xnf2sim is a utility that converts an unrouted XNF file to a Mentor EDDM single-object simulation model. This conversion can only be done on chip-level XNF files with EXT records, not on lower level modules embedded in a schematic. VHDL or Verilog simulation models can also be generated. Pld_xnf2sim is for functional simulation only.

Pld_men2edif

Pld_men2edif converts a Mentor schematic to a hierarchical EDIF netlist that is ready for implementation.

QuickHDL

QuickHDL™ (qhsim) is Mentor's simulator for behavioral VHDL, Verilog, or VHDL-based and Verilog-based gate-level designs composed of SIMPRIM elements.

QuickHDL PRO

QuickHDL PRO™ (qhpro) is Mentor's simulator for mixed schematic-based, VHDL-based, and Verilog-based designs. It can invoke QuickHDL to simulate HDL-based elements, or pld_quicksim to simulate Unified Schematic Library elements.

Pld_dsgnmgr

The Mentor Design Manager interface contains a Pld_dsgnmgr icon for the Xilinx Design Manager. Pld_dsgnmgr is the Xilinx Design Manager, which implements the design. You can access any individual Xilinx tool from the Xilinx Design Manager.

Pld_sg

Pld_sg is the Mentor schematic generator (SG), which creates a schematic from an EDDM single object netlist. You can use this tool to generate a schematic for the timing simulation netlist.

New Models for LogiBLOX Modules

You can enter a schematic using LogiBLOX symbols along with other Unified Libraries elements. For schematics, invoke LogiBLOX from within pld_da by using the Xilinx Libraries menu (**Libraries** → **Xilinx Libraries** → **Logiblox**). In addition, EDDM simulation models are automatically created for LogiBLOX symbols during symbol creation.

For VHDL or Verilog LogiBlox models, invoke LogiBlox from the pld_dmgr's tool window, or from the popup session window within pld_da.

EDIF

This release supports EDIF 2 0 0 for design implementation. Refer to the Xilinx EDIF specification for supported constructs.

Cross-Probing

Cross-probing is a way of cross-referencing between the original schematic and the timing simulation model after placement and routing. Once a Mentor design is translated, expanded, mapped, placed, and routed, you can extract the back-annotation information and create a hierarchical EDIF netlist. After you convert this EDIF to an EDDM model using `pld_edif2tim`, you submit it to `pld_dve` to create a viewpoint and then to `pld_quicksim` for timing simulation. The resulting data base preserves the design hierarchy, and although it is created in terms of the SIMPRIM library, most of the original net names are still available. You enable cross-probing by invoking QuickSim with the `-cp` option. This option invokes `pld_dve` as well as `pld_quicksim`. You then open the original design viewpoint in `pld_dve` and view the desired design sheet. If you display the original schematic in `pld_dve`, you can select nets on the original schematic and view them in the QuickSim trace window.

For more details on cross-probing, see the following sections:

- [“Cross-Probing” section of the “Schematic Designs” chapter](#)
- [“Performing Timing Simulation” section of the “Schematic Design Tutorial” chapter](#)
- [“Invoking Pld_quicksim” section of the “Schematic Design Tutorial” chapter](#).

Timing Simulation

This release supports back-annotated timing simulation after placement and routing. `Pld_edif2tim` translates the routed EDIF file to an EDDM single-object netlist.

Schematic Generator

The schematic generator is a utility that you can optionally use to generate a hierarchical schematic from a back-annotated EDDM model. This is not a required step since you can instead use cross-

probing with the back-annotated EDDM model and the original schematic for simulation without generating a back-annotated schematic. You can invoke the schematic generator from within the design manager or from a shell by typing `pld_sg`. You must have a Mentor schematic generator license in order to use this tool.

Timing Constraints

You can add timing constraints to the schematic as properties. You can also place them in a UCF (user constraints file) that NGDBuild can process. If a conflict arises between the timing information in the EDIF file and in the constraints file, the information in the constraints file prevails.

Design Flows

You use different PLD design flows for performing design entry, implementation and simulation depending on whether you use schematic design entry or HDL design entry.

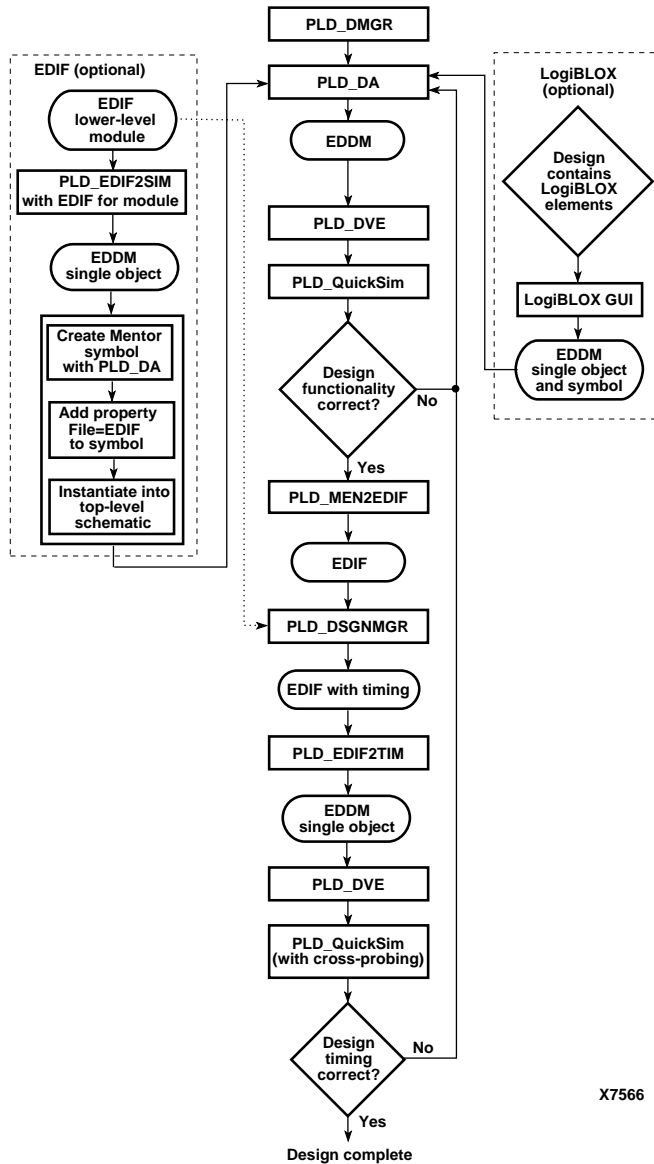
In either case, the easiest and most automatic way is to use the application icons in the Design Manager window. You can also run the various programs in the design flow manually from the UNIX shell. The shell commands are described in the [“Manual Translation” chapter](#).

The Mentor interface supports the following design flows:

- Schematic entry with the Unified Libraries components, Logi-BLOX symbols, or both
- Schematic entry with Unified Library components with some models expressed in Xilinx compliant EDIF or XNF
- Top-down HDL (Verilog/VHDL) design entry and synthesis
- Mixed schematic and VHDL design with VHDL on top
- Mixed schematic and VHDL design with schematic on top

Schematic Entry Design Flows

The schematic entry design flows are illustrated in the following three figures:



X7566

Figure 1-2 Schematic Design Entry Including EDIF-Based and LogiBLOX Modules

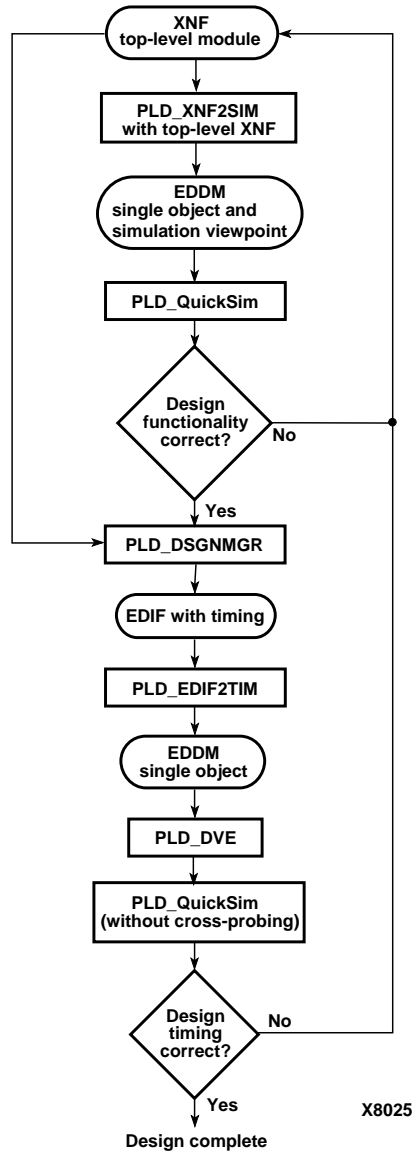


Figure 1-3 Design Entry with XNF Top-Level Module

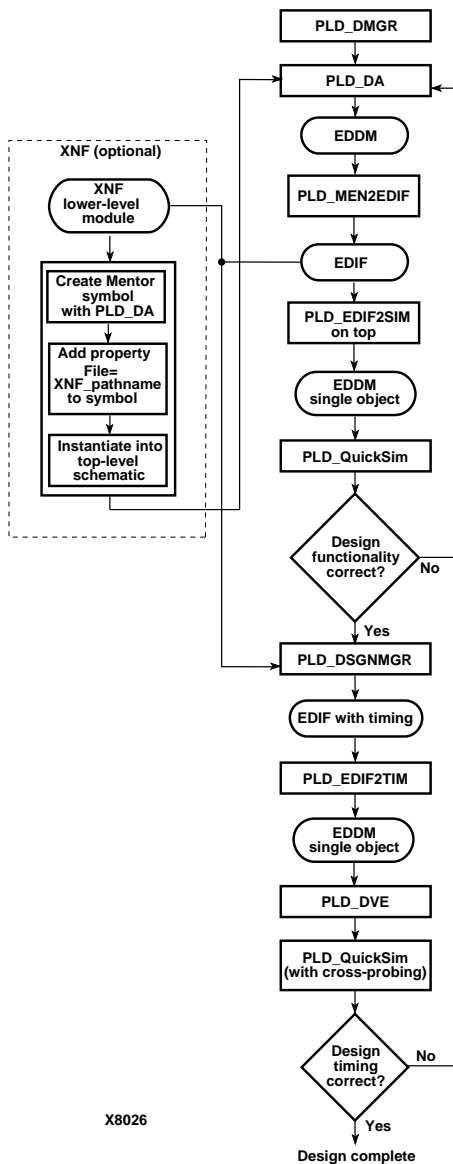


Figure 1-4 Schematic Design Entry with XNF Module

HDL Entry

The following figure shows the design flow for VHDL and Verilog design entry and synthesis for all supported technologies.

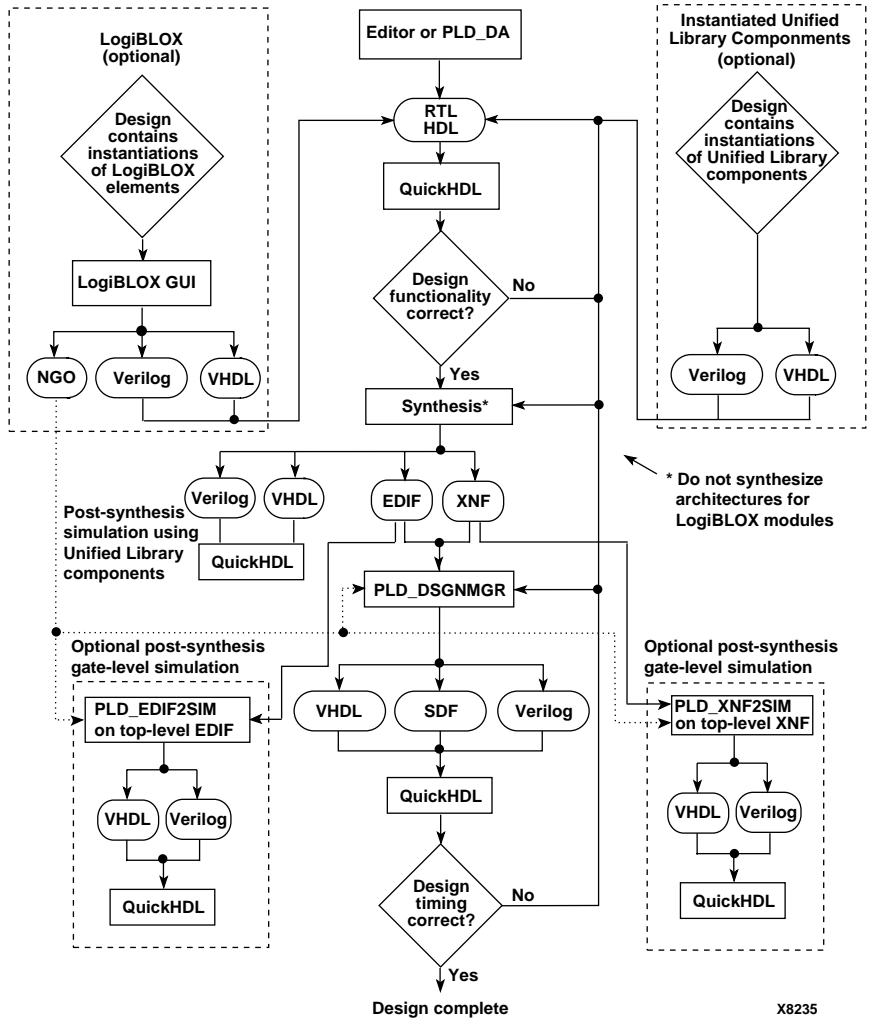
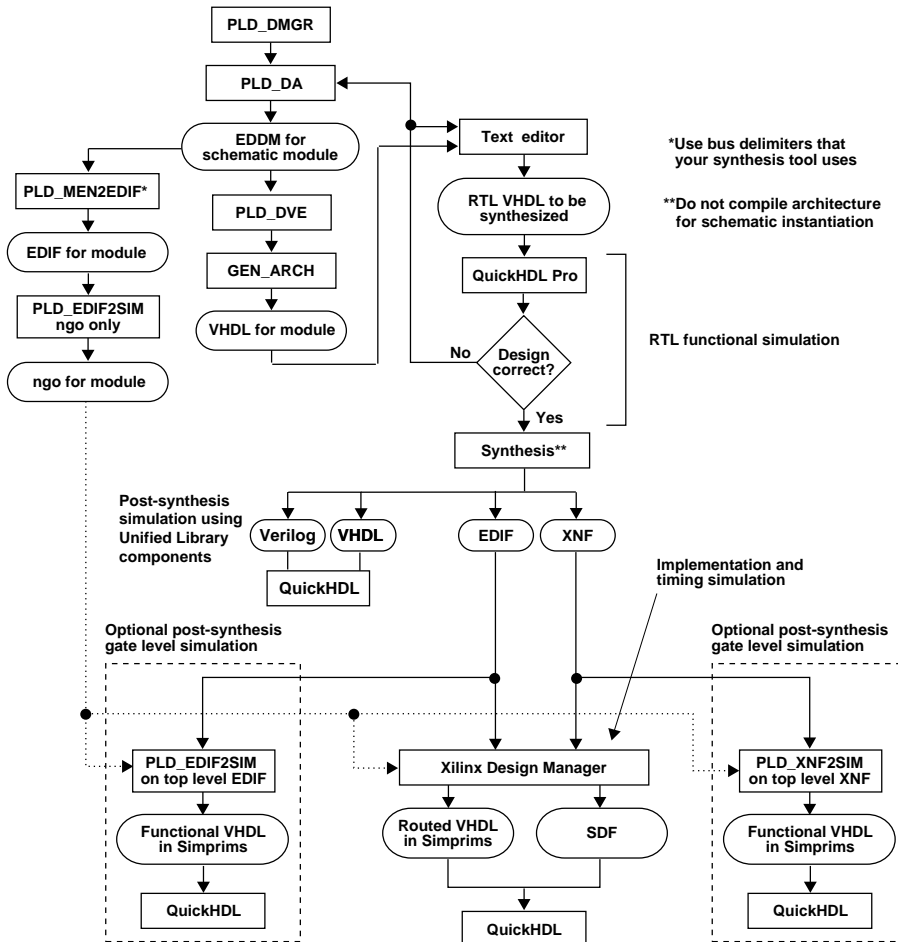


Figure 1-5 HDL (Verilog/VHDL) Design Entry and Synthesis

Mixed Schematic and VHDL Flow with VHDL on Top

The design flow for design entry of a top-level VHDL design with a schematic sub-module embedded within is illustrated in the following figure.



X8234

Figure 1-6 Mixed Schematic and VHDL Design with VHDL on Top

Mixed Schematic and VHDL Flow with Schematic on Top

The design flow for design entry using a mixture of schematics, VHDL, and Verilog is illustrated in the following figure.

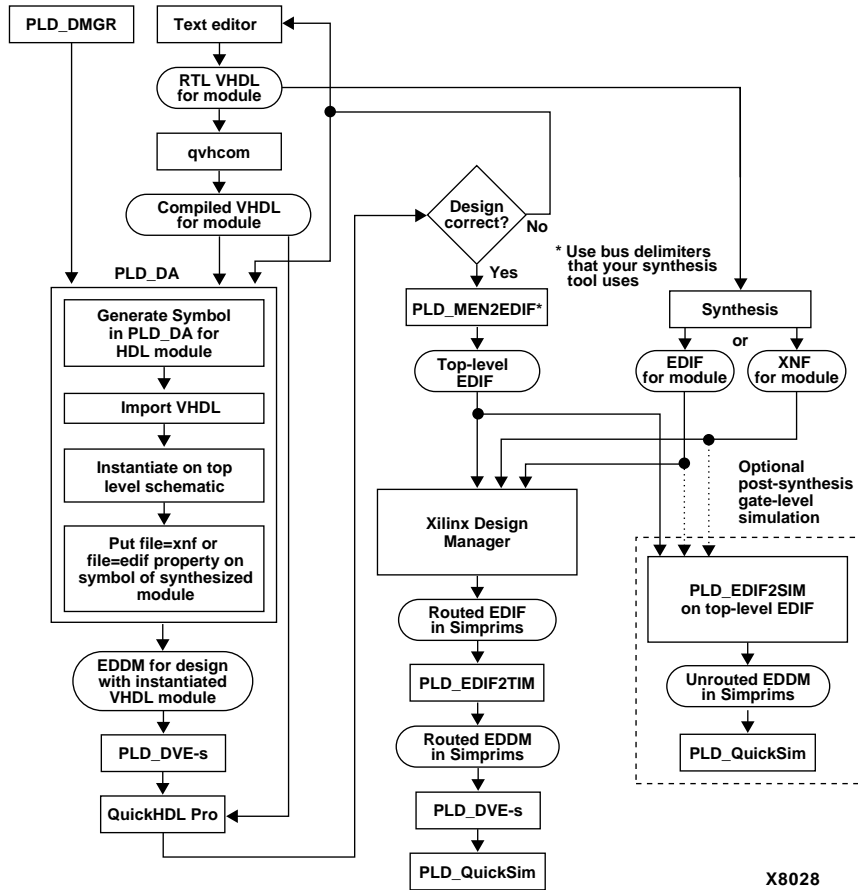


Figure 1-7 Mixed Schematic and VHDL Design with Schematic on Top

Inputs

The Mentor interface accepts netlists in EDIF or XNF format.

EDIF

You can submit an EDIF Level 2 0 0 netlist based on a design using Unified Libraries components. The following restrictions apply:

- Only the netlist and schematic types of EDIF are supported.
- Only one design per EDIF file is allowed.
- An EDIF file can contain one design component or multiple components. The EDIF2NGD utility converts each file to an NGO file. NGDBUILD uses a top-level NGO file, which refers to the other NGO files, to create the NGD file.

XNF

The Mentor interface can accept one of the following XNF netlists:

- An XNF netlist created by third-party netlist writers that meet the specifications of XNF version 6.1
- An XFF netlist created by XNFMerge version 6.1
- An XTF netlist created by XNFPrep version 6.1

An XNF netlist can represent all or part of a design. To be included in the netlist of a schematic design, a component must be tagged with the FILE property indicating the path name of the XNF file.

If a lower module is expressed in XNF, the top level must be run through EDIF2SIM in order to create a simulation netlist. The lower-level XNF file can not be run through XNF2SIM by itself since its lack of EXT records prevents XNF2SIM from knowing which signals should become module pins.

Outputs

The Mentor interface generates a back-annotated simulation netlist file based on the following:

- QuickPart-based SIMPRIM models and a flat/hierarchical EDIF netlist.

- VHDL-based SIMPRIM models, a structural VHDL netlist, and a SDF delay file.
- Verilog-based SIMPRIM models, a structural Verilog netlist, and a SDF delay file.

Files

The following Xilinx specific files are involved in processing a design through the Mentor interface:

- The EDN file is a post-route EDIF netlist file that expresses timing in SIMPRIM library elements instead of Unified Libraries elements.
- The NCD file contains a representation of the physical design.
- The NGA file contains physical timing delay information.
- The NGD file contains a logical design hierarchy expressed in the Xilinx implementation primitives.
- The NGM file contains a representation of the logical design. It also contains optimization information.
- The NGO file contains netlist information in a proprietary data base format; it is a binary file.
- The SDF file contains timing delay information.
- The V file contains the structural design based on Verilog-based SIMPRIM models.
- The VHD file contains the structural design based on VHDL-based SIMPRIM models.
- The XNF file is the Xilinx netlist format used prior to the use of EDIF in the current release. In the current Mentor Interface flow, XNF is only used as an import format option.
- The PCF file is the physical constraints file.
- The UCF file is the User Constraint File for specifying the user's timing and placement constraints for place and route.

Tutorials

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.

Online Help

The Mentor interface contains online help which is available from each application's dialog box. Help contains information about the Mentor features offered in the interface but does not contain information about the Xilinx features. The Mentor software is supplied with the BOLD_Browser, a set of online manuals. This online manual is the online help for the Xilinx features.

Getting Started

This chapter describes how to configure your system for the Mentor Graphics Design Manager, and how to invoke and exit the Mentor Graphics Design Manager. This chapter contains the following sections:

- “Configuring Your System” section
- “Invoking the Design Manager” section
- “Exiting the Design Manager” section

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:

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

Note: Path names of directories will vary. (For example, `$XILINX/bin/sol` would be `$XILINX/bin/hp` if you are running the Xilinx software on an HP workstation.) For more information on paths and environment variables, refer to the release notes that came with your software package.

`XILINX` is the directory where all Xilinx software is located.

`LCA` is the directory which includes Mentor-Interface files such as Xilinx libraries, translators, and scripts.

SIMPRIMS is the directory where the Mentor SIMPRIM models are located.

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.

- **MGLS_LICENSE_FILE**

This variable must point to a valid FlexLM license file that lists the Mentor Graphics license daemon and licensed software features, as supplied by Mentor Graphics. A sample license file may begin as follows:

```
SERVER tequiero 9542df17 1700
DAEMON mgcld /tools/mentor/lib/mgcld
      /usr/local/data/mentor.opt
FEATURE falconfw_s 8.0 31-dec-1997 10 ...
```

- **LD_LIBRARY_PATH**

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

```
setenv LD_LIBRARY_PATH $MGC_HOME/shared/
lib:$MGC_HOME/lib:$XILINX/bin/sol:/usr/openwin/lib
```

On HP workstations, leave out /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 or network. All components in designs created in the Mentor Graphics B.x environment reference the variable \$LCA, while back-annotated timing models

reference the variable `$$SIMPRIMS`. It is also important that the `$$LCA` and `$$SIMPRIMS` variables be instantiated, but not defined, in the file pointed to by `$$MGC_LOCATION_MAP`. With all these elements, the location-map file should, at a minimum, look like:

```
MGC_LOCATION_MAP_1
(empty line)
$$MGC_GENLIB
(empty line)
$$LCA
(empty line)
$$SIMPRIMS
(empty line)
```

The `MGC_LOCATION_MAP_1` line indicates that this is a version 1 location-map file. (You can also use the version `MGC_LOCATION_MAP_2`, which adds features such as outside file inclusion.) The three soft names with blank lines indicate that the Mentor Graphics software should pull the associated values from the parent environment.

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

- **MGC_WD (Optional)**

This variable should point to the working directory. You can have this variable always point to your current directory by setting it to “.”

Xilinx tools ignore the `MGC_WD` variable.

- **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, typically `$$XILINX/mentor/data`.

- **SIMPRIMS**

This points to the directory where Xilinx simulation models are located. This should be set to `$$LCA/simprims`.

Refer to the release notes for additional information on paths and environment variables.

Invoking the Design Manager

To invoke the Design Manager from the operating system, type `pld_dmgr`.

The Design Manager window appears, as shown in the “**Mentor Design Manager Window**” figure of the “Introduction” chapter.

Invoking Applications in the Design Manager

You can use either an icon or the Navigator to invoke an application from the Design Manager.

Tools Window Icons

To use an icon to open an application, double-click the left mouse button on the icon in the Tools window.

A dialog box appears that allows you to set options, or the application is executed.

Navigator Window

If you want to load a specific design, you can also use another method of invoking an application.

1. Select the design object in the Navigator window with the left mouse button, and press the right mouse button.
2. Select **Open** from the Navigator menu.
3. Select the appropriate application from the popup menu.

Only the applications that can be executed on the selected object will be displayed in the popup menu.

A dialog box appears that allows you to set options, or the application is executed.

Exiting the Design Manager

To exit the Design Manager, move the cursor to the title bar of the Design Manger window, press the right mouse button, and select **Quit** from the popup menu

Schematic Designs

This chapter describes how to use the Mentor Graphics Design Manager and Design Architect to design with pure schematic designs. It contains the following sections:

- “Design Flows” section
- “Design Entry” section
- “Functional Simulation” section
- “Implementing Schematic Designs” section
- “Timing Simulation for Schematic Designs” section

Design Flows

Three pure schematic design flows are shown in the “Schematic Entry Design Flows” section of the “Introduction” chapter. This chapter describes how to work with designs using the pure schematic design flows.

Design Entry

Invoking Design Architect

You can use either the `p1d_da` icon or the Navigator to invoke Design Architect from the Design Manager.

To invoke Design Architect with the `p1d_da` icon in the Tools Window:

1. Double-click the left mouse button on the `p1d_da` icon.

A Design Architect window similar to that shown in the “Design Architect Window” figure appears but without displaying a sche-

matic. You can use the Open Sheet icon in the Session Palette to open a schematic sheet.

If you want to load a specific design, you can invoke Design Architect from the Navigator as follows:

1. Select the design in the Navigator window and press the right mouse button.
2. Select **Open** → **p1d_da** from the Navigator pop-up menu.

A Design Architect window similar to that shown in the following figure appears.

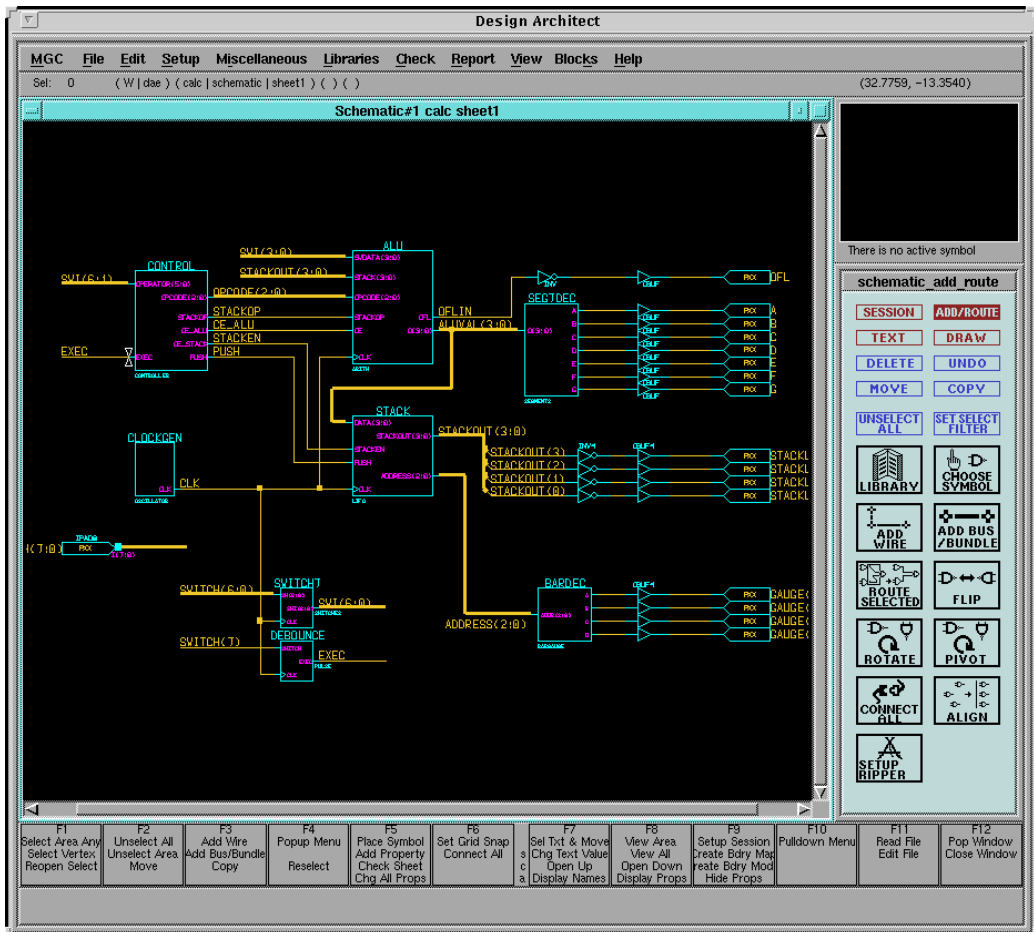


Figure 3-1 Design Architect Window

Exiting Design Architect

To exit Design Architect, move the cursor to the title bar of the Design Architect window, press the right mouse button, and select **Quit** from the popup menu.

Loading a Schematic

If a design is not loaded into the schematic window, the Session Palette (session_palette) appears on the right-hand side of the Design Architect window. If one design is presently loaded and you want to also load another design, click on the Session icon in the schemataic_add_route Session Palette.

To load an existing schematic into the Design Architect window, follow these steps.

1. Click on the **Open Sheet** icon in the Session Palette.

The Open Sheet dialog box appears, as shown in the following figure.

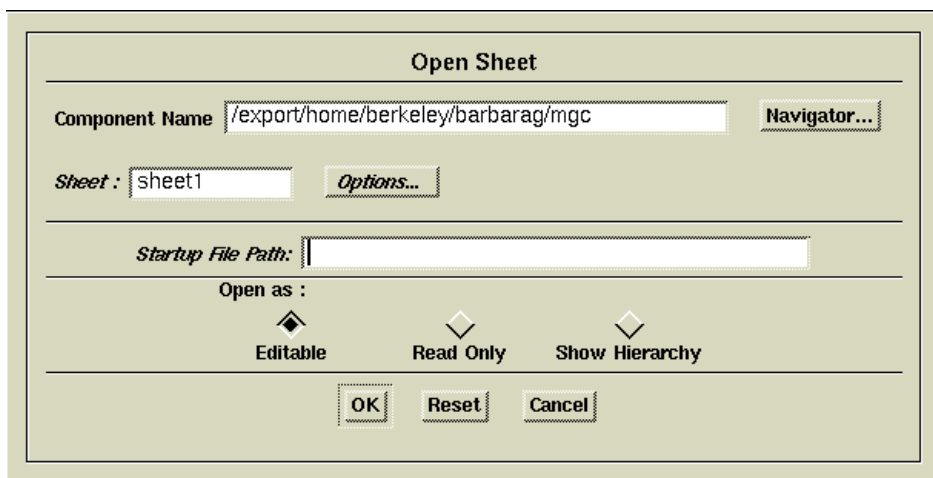


Figure 3-2 Open Sheet Dialog Box

2. To find an existing design, type the path and name of the component or schematic in the Component Name field, or click on **Navigator** to find it.

Note: If the component has not yet been created, open pld_da in the Tool Window. Then open a sheet from the Session Palette. In the Open Sheet dialog box, assign the component a name and click OK.

3. In the Sheet field, type the name of the schematic sheet that you want to display.
4. In the Open As field, select **Editable**.

5. Click on **OK**.

The schematic sheet now appears in the Design Architect window. The schematic number, name of the design, and sheet number appear in the title bar. The Session Palette changes to the Schematic Palette (schematic_add_route).

Creating the Design Component

When you save your schematic in Design Architect, the following items are created:

- A *design*.mgc_component.attr file
- A *design* component directory

The *design* component directory may contain schematic files, symbol files, and viewpoint files. The *design* directory and the *design*.mgc_component.attr file together are known as a Mentor component object.

Adding Components

Adding Xilinx library Components

1. To add a component from the Xilinx libraries, select **XILINX Libraries** from the Libraries pull-down menu.
2. In the Schematic Palette, click on the desired technology library.

Note: You cannot mix old XC4000EX library components with XC4000X library components. Use Convert Design to convert XC4000EX designs to XC4000X before instantiating new XC4000X library components.

3. Click on **BY TYPE** to select a category of element, or **ALL PARTS** to select a specific element.
4. Click on the desired element, move the cursor to the desired location on the schematic, and click on the left mouse button to place it.

Xilinx Libraries

In Design Architect, the Xilinx Libraries menu contains the Unified 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. Refer to the *XACT Libraries Guide* for detailed information on the Xilinx Libraries.

Primitives and Macros

The Xilinx Libraries contain the following types of components:

- **Primitives:** These are pads and basic logic elements, such as gates, latches, flip-flops, buffers, and oscillators.
- **Soft macros:** These are schematics that contain primitives and other soft macros. Soft macros have pre-defined functionality and often have fixed mapping, placement, and routing to provide the most efficient use of resources and the fastest speed.

LogiBLOX

LogiBLOX allows you to synthesize common data functions such as addition, that are optimized for a particular family. Refer to the *LogiBLOX User Guide* for information on LogiBLOX components.

Using the Xilinx Libraries

The following procedure describes selecting a component from the Unified Libraries and placing it in your schematic. Do not mix components from different technologies (families).

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

1. Select **XILINX Libraries** from the Libraries pull-down menu. The schematic palette is replaced by the Xilinx libraries menu palette.
2. 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.

3. Select a component from the library list.
4. Move the cursor into the schematic window. An outline of the selected component appears.
5. 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.

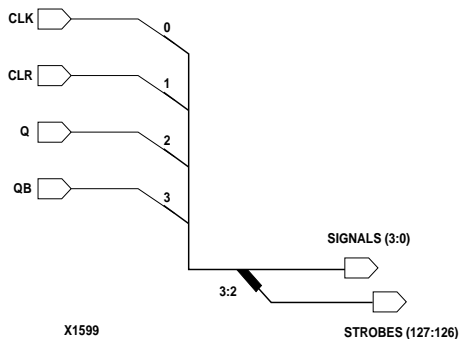


Figure 3-3 Bus Ripper

In Mentor, there are two types of bus rippers, implicit and explicit. An explicit ripper uses the `RULE` property to specify the index. The `RULE` property lets you specify a name for the net. An implicit ripper does not have a rule property and the name of the net must be the same as the name of the bus.

To add a bus ripper to a bus, perform the following procedure:

1. If you don't already have a bus in your system, add one and give it a name such as ADDR (31:0).
2. Draw a net to the bus.
3. In the Choose Bus Bit dialog box that opens, specify the bit number of the net that you want to rip.

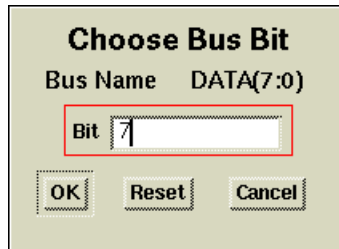


Figure 3-4 Choose Bus Bit Dialog Box

4. Design Architect automatically inserts a ripper which by default is implicit.
5. To specify a non-implicit ripper, open the Setup Ripper dialog box by doing one of the following:
 - Select the **Setup** → **Ripper** menu
 - Select the Setup Ripper icon in the Schematic palette

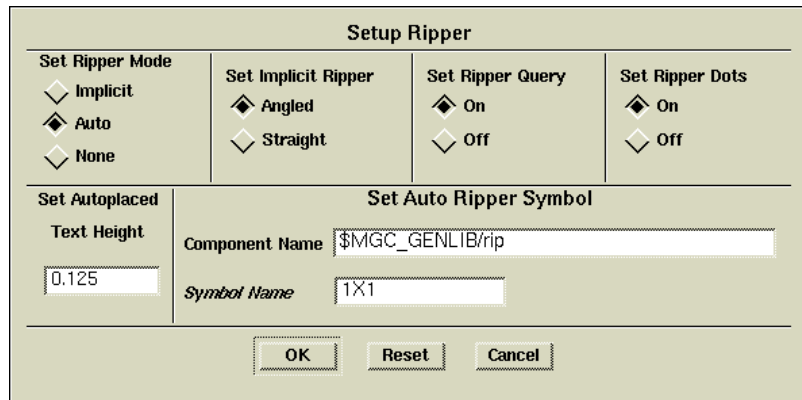


Figure 3-5 Setup Ripper Dialog Box

6. In the Setup Ripper dialog box, Select the Auto ripper mode and click OK.
7. Specify the bit in the Choose Bus Bit dialog box and the bus name if it is not already named.

An explicit 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 popup 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 **“Schematic Design Tutorial”** chapter in this Manual and the *Design Architect User’s Manual*.

Adding Properties

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 you design 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. They provide information to the implementation tools during the processing of your schematic design.

Properties

This section describes the properties that are unique to Mentor or that are required when working with Xilinx PLDs using Mentor.

Properties, or attributes, are instructions placed on symbols or nets in an FPGA or CPLD schematic that allow you to control aspects of software processing. They express information specific to each design, unlike run-time options entered in the Xilinx Design Manager.

This section describes the properties that are unique to Mentor schematics or that are required. The *Xilinx Libraries Guide* describes the other attributes that you can place on a Mentor schematic.

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. When adding PINTYPE properties, select **PINTYPE** from the list of properties and type **in**, **out**, or **ioxo** for input, output, or bidirectional, respectively, in the value box.

INST

Use the INST property to uniquely identify an instantiation of a component or symbol in a design. Design Architect assigns a default INST property to the symbol of each instantiation (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.

Do not place the COMP property on user symbols since COMP indicates that the symbol is a Xilinx library primitive.

CYMODE

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

INTERNAL

Use the INTERNAL property to identify unbonded IOBs.

Adding Properties

Use the following procedure to add properties to instances, pins, or nets.

1. Select the instance, pin, or net.

If you are applying a property to an instance, select the instance. Be sure nothing else is selected.

If you are applying a property to a net, select the vertex where the output of a symbol connects to the net. Be sure you have selected only that vertex; a single star should appear at that location.

2. Press the right mouse button.

The Instance popup menu should appear if you have selected an instance. The Net popup should appear if you have selected a net or a pin. If the Mixed Selection popup appears instead, you have more than one design object selected. Choose **Unselect** → **All**, then select the instance or net and try pressing the right mouse button again.

3. Select the **Properties** → **Add** → **Add Single Property** command from the popup menu.

The Add Property dialog box appears, as shown in the following figure.

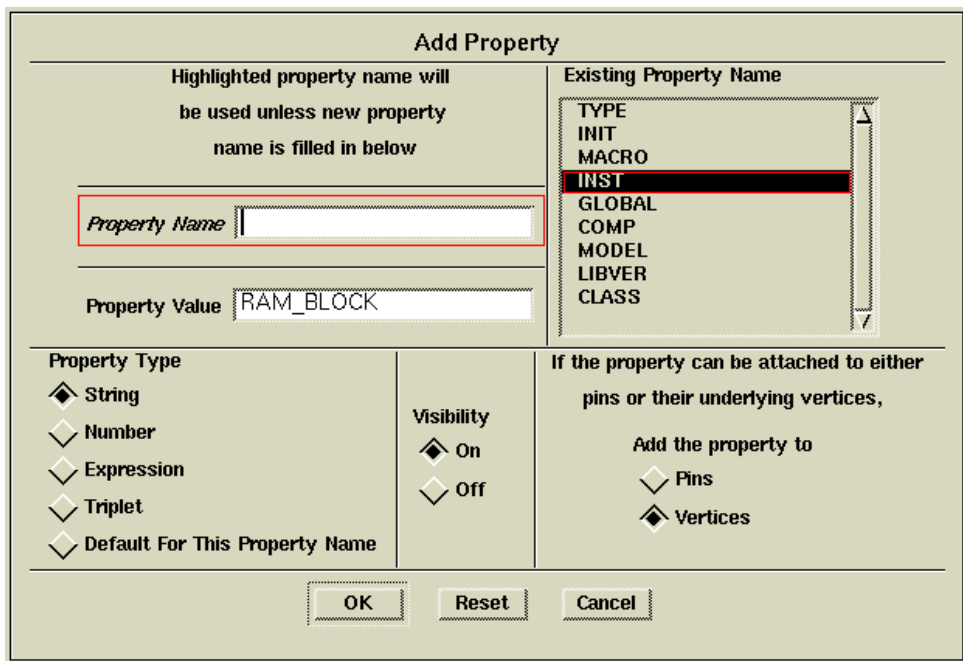


Figure 3-6 Add Property Dialog Box

4. In the Property Name box, type the name of the property, for example, OPT, or click on it in the Existing Property Name list.
5. Type the value, for example, OFF, in the Property Value box.

Note: For some properties, the property name and the property value are identical.

6. Because most properties take strings, select **String** in the Property Type field.
7. In the Visibility field, select **On** if you want the property to be visible.
8. In the field asking whether to attach the property to pins or vertices, select **Vertices** if you are attaching it to a net or an instance. Select **Pins** if you are attaching it to a pin.
Body, pin, and net properties are always added to vertices.
9. Select **OK**.

The ADD PR prompt bar appears.

10. Position the cursor where you want to place the property, usually above the component or net.
11. Click the left button to place the property.

Adding the Net Property to Nets

Use the Net property to label signals in your design. To add a Net property, you can either follow the instructions in the “**Adding Properties**” section or follow these steps:

1. Select all of the nets that you want to name.

Unlike the procedure in the “**Adding Properties**” section, it is not necessary to select a single vertex for each net.

2. Press the right mouse button.

The Net popup appears. If the Mixed Selection popup appears instead, select **Other Menus**, then select **Net Menu**.

3. Select **Name Nets** from the popup menu.

The ADD PR bar appears.

4. Type the property value in the Property Value box.
5. Click **OK**.
6. Position the cursor where you want to place the property, usually above the component or net.
7. Click the left mouse button to place the property.
8. Repeat the netname entry for each net you have selected.

Modifying Property Values

You can only modify property values, not property names.

To modify a property’s values, perform the following steps:

1. Select the entity whose property value you want to change.
2. Press the right mouse button to display the popup menu.
3. Select **Properties** → **Modify**.

A dialog box appears listing the properties of the selected object. The following figure shows an example.

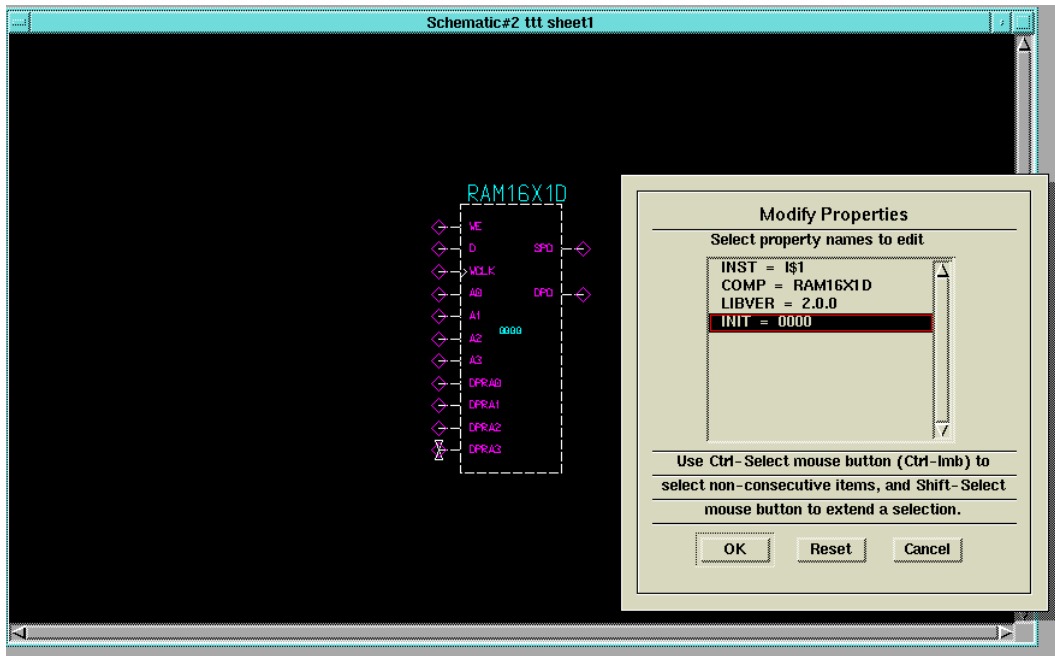


Figure 3-7 Modify Properties Dialog Box

4. Select the property that you want to modify.
5. Click on **OK**.

The Modify Property dialog box is displayed, as shown in the following figure.

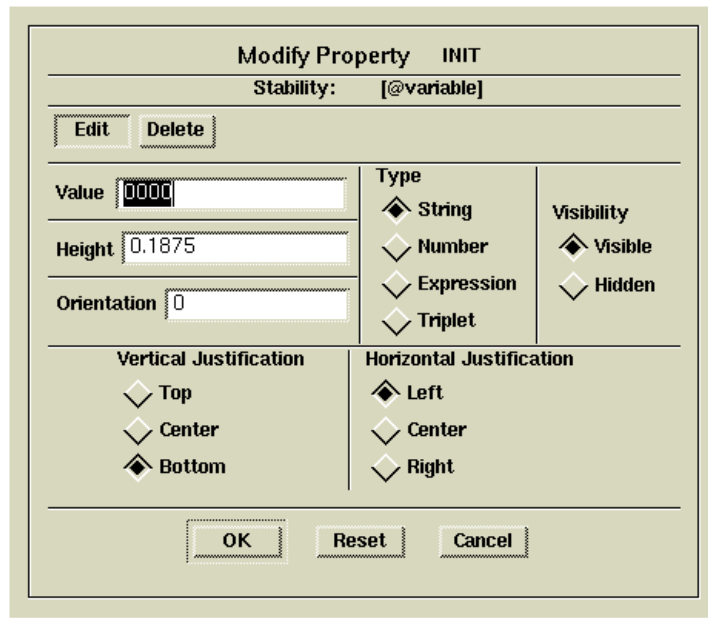


Figure 3-8 Modify Property Dialog Box

6. Type the new value in the Value field.
7. Set any other options. Most of the time the default settings are appropriate.
8. Click on OK.

Entering Timing Specifications

The Mentor netlist writer program (ENWRITE) converts all property names to lowercase letters, and the Xilinx netlist reader EDIF2NGD then converts the property names to uppercase letters. To ensure references from one constraint to another are processed correctly, observe these guidelines:

- A *TSidentifier* name should contain only uppercase letters on a Mentor Schematic (TSM_{MAIN}, for example, but not TS_{main} or TS_{Main}).
- If a *TSidentifier* name is referenced in a property value, it must be entered in uppercase letters. For example, the TSID1 in the

second constraint below must be entered in uppercase letters to match the TSID1 name in the first constraint.

```
TSID1 = FROM: gr1: TO: gr2: 50;  
TSMAIN = FROM: here: TO: there: TSID1: /2;
```

Creating New Groups from Existing Groups

The Mentor netlist writer program (ENWRITE) converts all property names to lowercase letters, and the Xilinx netlist reader EDIF2NGD then converts the property names to uppercase letters. To ensure references from one constraint to another are processed correctly, observe these guidelines:

- Group names should contain only uppercase letters on a Mentor Schematic (MY_FLOPS, for example, but not my_flops or My_flops).
- If a group name appears in a property value, it must also be expressed in uppercase letters. For example, the GROUP3 in the first constraint below must be entered in uppercase letters to match the GROUP3 in the second constraint.

```
TIMEGRP GROUP1 = gr2: GROUP3;  
TIMEGRP GROUP3 = FFS: except: grp5;
```

Functional Simulation

Functional simulation provides an effective means of 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.

You can functionally simulate XNF or EDIF based designs by using `pld_xnf2sim` or `pld_edif2sim` to convert the designs to a Mentor simulation model. The EDIF design must be Xilinx compatible and expressed in Unified Library components. The following figure illustrates the design flow for these types of designs.

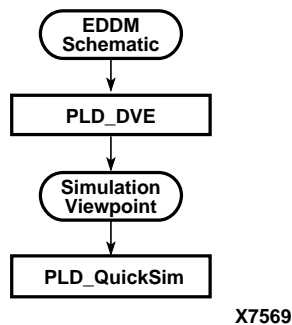


Figure 3-9 Functional Simulation Flow Diagram

The “[Performing Functional Simulation](#)” section of the “[Schematic Design Tutorial](#)” chapter in this manual provides a detailed example of the steps involved in functional simulation.

Simulating Pure Schematic Designs

This section describes how to simulate purely schematic designs—designs that are composed solely of elements from the Unified Libraries and that have been entered through Design Architect. Performing functional simulation on a pure schematic design consists of creating a viewpoint in `pld_dve` from the schematic that you created in Design Architect and using `pld_quicksim` to simulate the design.

Creating the Viewpoint

After creating a schematic design with Design Architect and a Xilinx library, the next step in the functional simulation flow is to configure a viewpoint for the simulator. Without a correct simulation viewpoint, you will not be able to simulate your design. The viewpoint defines primitives and parameters for design evaluation and analysis.

`Pld_dve` invokes the Mentor Graphics Design Viewpoint Editor (DVE) to configure a viewpoint for Xilinx designs.

Create the viewpoint for the top-level component that was created in Design Architect.

1. To invoke DVE, double-click the left mouse button on the `pld_dve` icon in the Design Manager Tools window.

Alternatively, you can select the top-level component in the Navigator window and click the right mouse button to invoke `pld_dve`.

The dialog box shown in the “**Pld_dve Dialog Box**” figure appears. For a more detailed description of DVE, refer to the *Mentor Graphics Design Viewpoint Editor Users Manual* and *Reference Manual*.

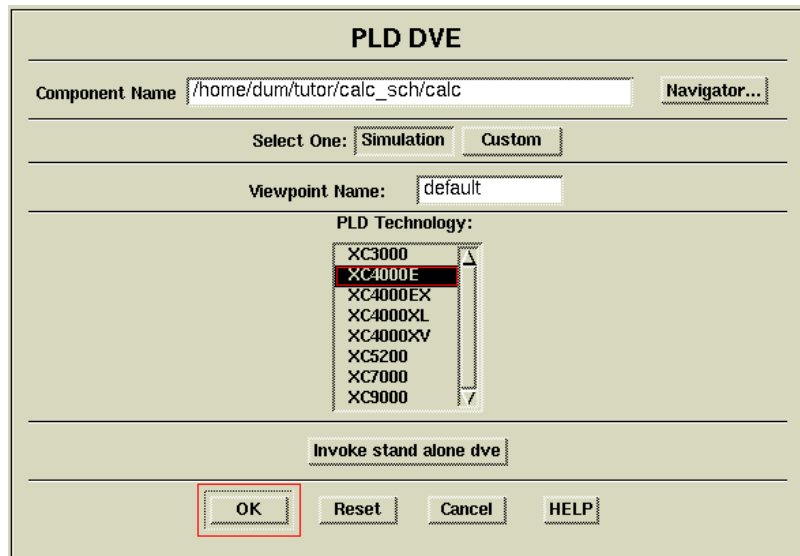


Figure 3-10 Pld_dve Dialog Box

2. Enter the design name in the Component Name field, or click on **Navigator** to browse a list of design names. If you invoked `pld_dve` from the Navigator window, the component is already selected.

If you click on the Navigator, you can select the component name, and the corresponding viewpoint name will appear in the Viewpoint Name field.

3. In the Select One field, select **Simulation**.

Select **Custom** if you want to open the selected viewpoint in DVE so that you can interact with it rather than accept the `pld_dve` default. Selecting custom invokes Mentor's DVE and opens the named viewpoint. You could use this to select a different model for a specific sub-module.

4. In the Viewpoint Name field, you can enter the viewpoint name if you do not want to use the default viewpoint.
5. In the PLD Technology field, select a technology.
6. Click on **Invoke Stand-Alone DVE** only if you want to invoke DVE to interact with Mentor's user interface instead.

This command brings up the DVE window to allow you to customize the viewpoint. For information on customizing a viewpoint, see the Mentor Graphics DVE user documentation.

7. Select **OK** to start `pld_dve`.

`Pld_dve` now generates a viewpoint with the same name as that entered in the Viewpoint Name field. It is in the format *component_name/viewpoint_name*.

You can also access `pld_dve` from a UNIX shell.

If you are converting a top-level XNF or EDIF netlist with `pld_xnf2sim` or `pld_edif2sim`, the simulation viewpoint is created for you automatically.

Simulating the Design

After creating the viewpoint, you can submit pure schematic designs to `pld_quicksim` for functional simulation.

1. To invoke `pld_quicksim`, double-click the left mouse button on the `pld_quicksim` icon in the Design Manager Tools window.

Alternatively, you can select the top-level component in the Navigator window and click the right mouse button to invoke `pld_quicksim`.

The PLD_QuickSim II dialog box, shown in the “**PLD_QuickSim II Dialog Box**” figure, appears on the screen. For more detailed information on the dialog box options, refer to the Mentor Graphics QuickSim documentation.

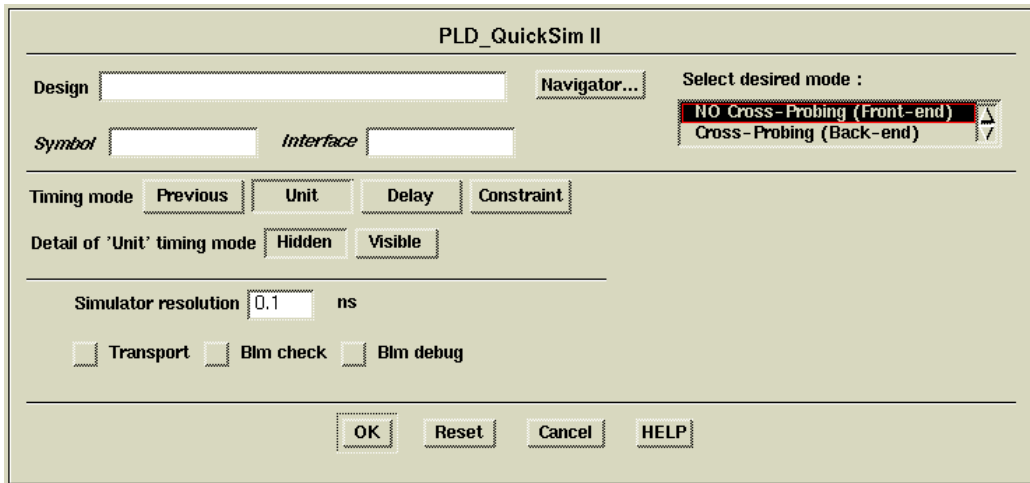


Figure 3-11 PLD_QuickSim II Dialog Box

2. In the Design field, enter the design name. If you selected the component in the Navigator window, the design name is already set.
3. In the Select Desired Mode box, click on **No Cross-Probing**, if it is not already selected (This is the default setting).

You can only select cross-probing for timing simulation for schematic designs, not for functional simulation. See the “**Cross-Probing**” section for more details about cross-probing.

4. In the Timing Mode field, select **Unit** for functional simulation.
5. In the Detail of “Unit” Timing Mode field, click on **Hidden**.
6. In the Simulator Resolution box, enter the smallest unit of time that you want to be visible in the simulator.

The smallest resolution allowed for Xilinx designs is 0.1 ns.

7. Click on **OK**.

Pld_quicksim now starts, and the QuickSim II window appears. The QuickSim II window functions as a waveform viewer; you can bring up the schematic and view the signals, or you can view the waveforms generated by the simulation. Consult the Mentor Graphics documentation for more information on how to view waveforms in this window.

Simulating Schematic Designs with LogiBLOX Elements

LogiBLOX creates a simulation model for LogiBLOX elements. However, you must still create a viewpoint on the top-level design with pld_dve before functionally simulating the design. Follow the instructions in “[Creating the Viewpoint](#)” section of the “[Simulating Pure Schematic Designs](#)” section in this chapter. Then submit the design to pld_quicksim, following the procedure given in the “[Simulating the Design](#)” section of the “[Simulating Pure Schematic Designs](#)” section in this chapter.

Simulating Schematic Designs with XNF Elements

To functionally simulate a pre-route XNF design, follow the steps in this section. The steps are illustrated in the following figure.

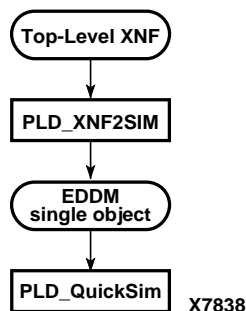


Figure 3-12 XNF Functional Simulation Flow

Creating the Design Component

Create the top-level design component as described in the “**Creating the Design Component**” section in this chapter. This provides an “anchor” for the converted design.

Converting the XNF File

The next step is to convert the XNF file to a simulation model.

1. In your schematic, create a symbol for each XNF element in your design.
2. Attach a FILE=*xnf_file_pathname* property to each symbol.
3. Double-click the left mouse button on the *pld_xnf2sim* icon in the Design Manager Tools window.

The resulting dialog box is shown in the following figure.

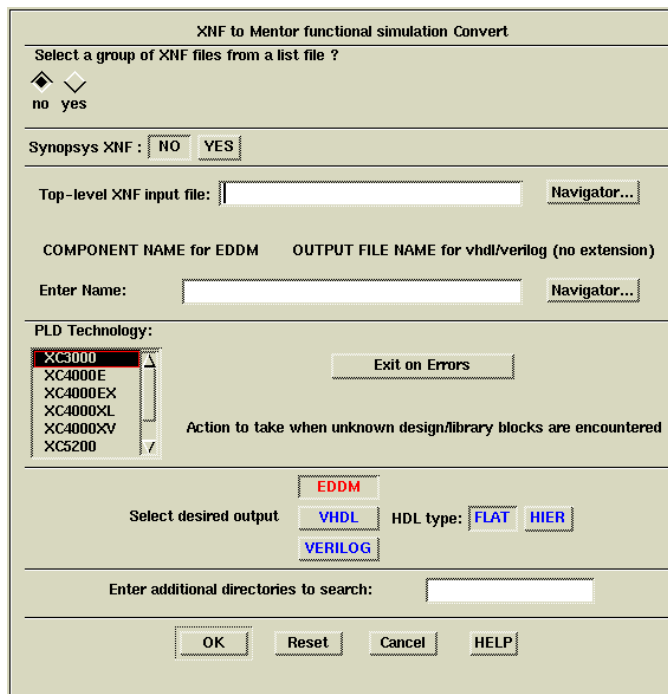


Figure 3-13 PLD XNF to Mentor Convert Dialog Box

Pld_xnf2sim uses all supporting XNF files from the directory in which the top-level XNF input file was submitted.

4. If the required XNF files are not in that directory, click **Yes** in the field asking “Select a group of XNF files from a list file?” and specify the path name of a file that lists the path names of all needed XNF files. Each path name is specified on a separate line in this file, for example:

```
/x/y/z/abc.xnf  
/x/y/z/def.xnf
```

5. In the Synopsys XNF field, select **No** if the XNF does not come from Synopsys.
6. In the Top-level XNF Input File field, type the name of your top-level XNF file, or click on **Navigator** to find it.
7. In the Enter Name field, enter the name of the symbol that you created in step 1, or click on **Navigator** to find it.

Note: If the symbol has not yet been created, a Mentor component is created with an EDDM-single-object model. At this point, you can use Design Architect to create a symbol for it. Click on **Open Symbol** and specify the path name of this component. A symbol is automatically created. Check the symbol, add the file=*xnf_file_pathname* property, and save it if the XNF file represents the entire design (if it has EXT statements for IO pins.). However, if the XNF does not contain EXT statements, you must manually create the symbol and assign the pins. In this case, the simulation model (EDDM_single_object) created by pld_xnf2sim will not correspond with this symbol, and functional simulation must be done by converting the entire design to EDIF and submitting the EDIF to pld_edif2sim to create a top-level component and use pld_quicksim to simulate. This top-level component and all its submodules will be expressed in terms of SIMPRIM primitives rather than the Unified Library components used for design entry.

8. In the PLD Technology field, select the appropriate architecture.
9. Leave the Exit on Errors button enabled if you want the program to exit when it encounters an unresolved block. Otherwise, click on the Exit on Errors button and it changes to Continue (Ignore Errors).
10. In the Select Desired Simulation Model field, select **EDDM**.

11. In the “Enter additional directories to search” field, enter all the directory pathnames that the program should search to find EDIF, XNF, and NGO files that define blocks in your design that have the File property on them.
12. Click OK.

This procedure produces a single-object simulation model for the specified symbol component.

Creating the Viewpoint

If you are converting a top-level XNF or EDIF netlist with `pld_xnf2sim` or `pld_edif2sim`, the simulation viewpoint is created for you automatically.

Simulating the Design

The rest of the simulation procedure is the same as that described in the “[Simulating the Design](#)” section of the “[Simulating Pure Schematic Designs](#)” section earlier in this chapter.

Simulating Schematic Designs with EDIF Elements

To functionally simulate a pre-route EDIF design, follow the steps in this section. The steps are illustrated in the following figure.

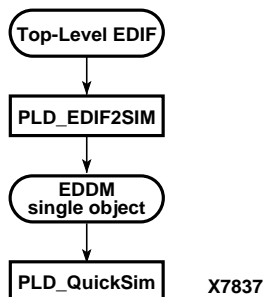


Figure 3-14 EDIF Functional Simulation Flow

Creating the Design Component

Create the top-level design component as described in the “[Creating the Design Component](#)” section in this chapter. This provides an “anchor” for the converted design.

Converting the EDIF File

The next step is to convert the EDIF file to a simulation model.

1. In your schematic, create a symbol for each EDIF element in your design.
2. Attach a FILE=*edif_file_pathname* property to each symbol.
3. Double-click the left mouse button on the pld_edif2sim icon in the Design Manager Tools window.

The resulting dialog box is shown in the following figure.

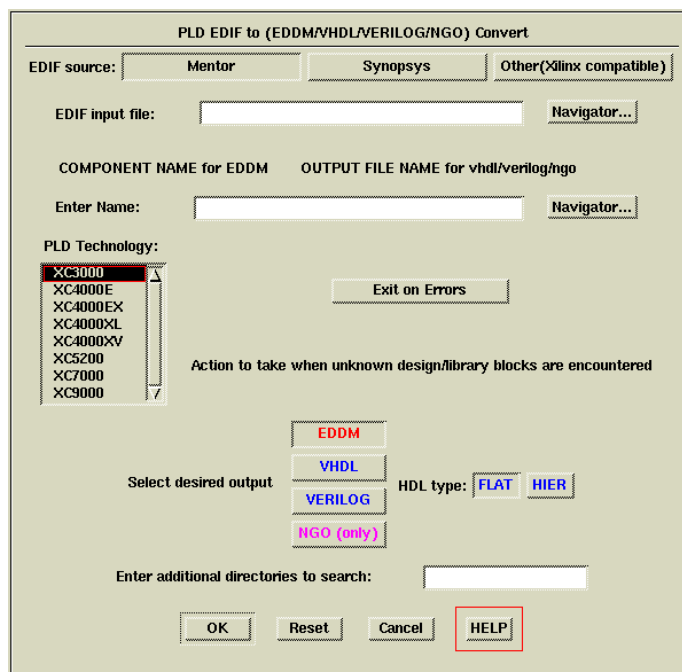


Figure 3-15 PLD EDIF to Mentor Convert Dialog Box

Pld_edif2sim uses all supporting EDIF files from the directory in which the top-level EDIF input file was submitted.

4. In the EDIF source field, select **Mentor**, **Synopsys**, or **Other** to specify the source from which the EDIF was generated. Specify **Other** if the EDIF comes from a vendor other than Mentor or Synopsys. When selecting **Other**, you must ensure that the EDIF is compatible with Xilinx EDIF.
5. In the Top-level EDIF Input File field, type in the name of your top-level EDIF file, or click on **Navigator** to find it.
6. In the Enter Name field, enter the name of the symbol that you created in step 1, or click on **Navigator** to find it.

Note: If the symbol has not yet been created, a Mentor component is created with an EDDM-single-object model. At this point, you can use Design Architect to create a symbol for it. Click on **Open Symbol** and specify the path name of this component. A symbol is automatically created. Check the symbol, add the file=*edif_file_pathname* property, and save it.

7. In the PLD Technology field, select the appropriate architecture.
8. Leave the Exit on Errors button enabled if you want the program to exit when it encounters an unresolved block. Otherwise, click on the Exit on Errors button and it changes to Continue (Ignore Errors).
9. In the Select Desired Output field, select **EDDM**.
10. In the “Enter additional directories to search” field, enter all the directory pathnames that the program should search to find EDIF, XNF, and NGO files that define blocks in your design that have the File property on them.
11. Click **OK**.

This procedure produces a single-object simulation model for the specified symbol component.

If you are converting an EDIF with pld_edif2sim, the simulation viewpoint is created for you automatically.

Simulating the Design

The rest of the simulation procedure is the same as that described in the “[Simulating the Design](#)” section of the “[Simulating Pure Schematic Designs](#)” section earlier in this chapter.

Implementing Schematic Designs

Once you complete functional simulation for schematic designs, you are ready to implement your design. You perform implementation with the Xilinx Design Manager, `pld_dsgnmgr`, which you invoke from the Mentor Design Manager or from a UNIX shell.

`Pld_dsgnmgr` first translates the design into a flattened or hierarchical netlist, then optimizes, places, and routes the design. It creates delay data for timing simulation and physical (bitstream) design data for downloading.

Design entry of pure schematic designs, or schematic designs with LogiBLOX elements, EDIF sub-modules, or XNF sub-modules produces an EDDM file that you must convert to EDIF with the `pld_men2edif` utility before implementing the design with `pld_dsgnmgr`. The following figure shows the design flow involved in implementing a design.

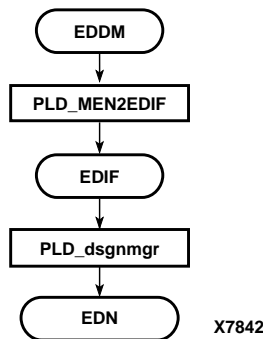


Figure 3-16 Design Implementation

Converting the EDDM Design

To convert your design to EDIF, follow these steps.

1. In the Mentor Design Manager, double-click the left mouse button on the `pld_men2edif` icon.

The dialog box shown in the following figure appears.

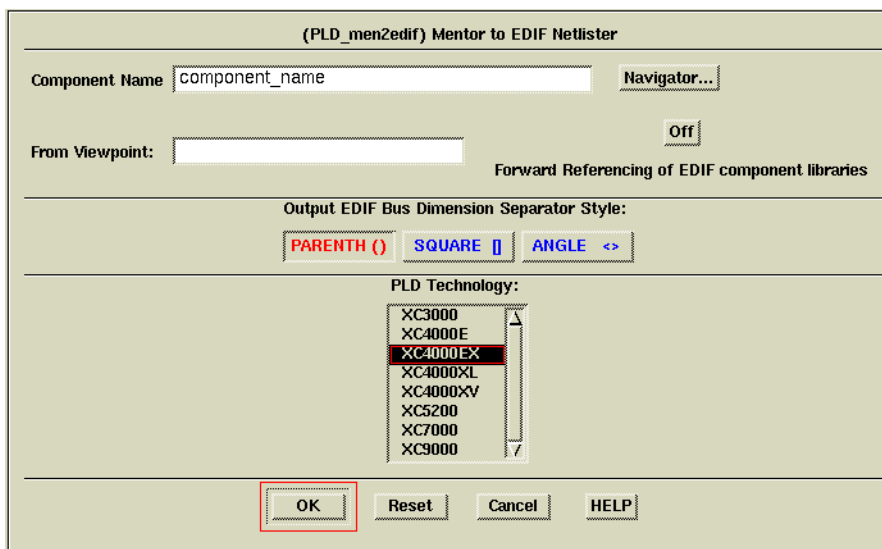


Figure 3-17 Mentor to EDIF Netlist Dialog Box

2. In the Component Name field, enter the component name, or click on **Navigator** to browse a list of design names.
3. In the From Viewpoint field, you can enter the viewpoint name if you do not want to use the default viewpoint. Alternatively, in step 2 you can select a viewpoint under the component.
4. Select the appropriate architecture for your design in the PLD Technology field.
5. Select the appropriate Bus Dimension Separator Style.
This is important if you are merging components from other design-entry tools into a single design. Choosing a bus-index delimiter lets you insure that the bus-index delimiters that `pld_men2edif` writes out are consistent with those of any other design-entry tools with which you are interfacing. Mentor EDIF uses parentheses. Synopsys EDIF uses angle brackets.
6. Click **OK**.

Pld_men2edif now produces an EDIF file that you can submit to the Xilinx Design Manager, pld_dsgnmgr. The output name is *component_name.edif*.

Implementing the Design

The Xilinx Design Manager is a graphical design-flow and project manager. The Xilinx Design Manager takes your design, represented by the EDIF file from pld_men2edif, and implements it in an FPGA or CPLD. You can also use the Xilinx Design Manager to generate timing information that you can import into QuickSim or QuickHDL.

The Xilinx Design Manager, pld_dsgnmgr, can accept an EDIF file, or if your design is a pure XNF design, it can accept an XNF file.

For a more in-depth discussion of the flow, including advanced implementation options, see the *Core Tools Reference Guide*.

To implement your design follow these steps:

1. Within the Mentor Design Manager, select the EDIF icon for your design in the Navigator, then select **Right Mouse Button** → **Open** → **pld_dsgnmgr**. The Xilinx Design Manager appears as shown in the “**Xilinx Design Manager**” figure. The tool automatically creates a Xilinx project called *your_design_name*. Xilinx project information is kept in a file called *xproject/your_design_name.prj* by default.

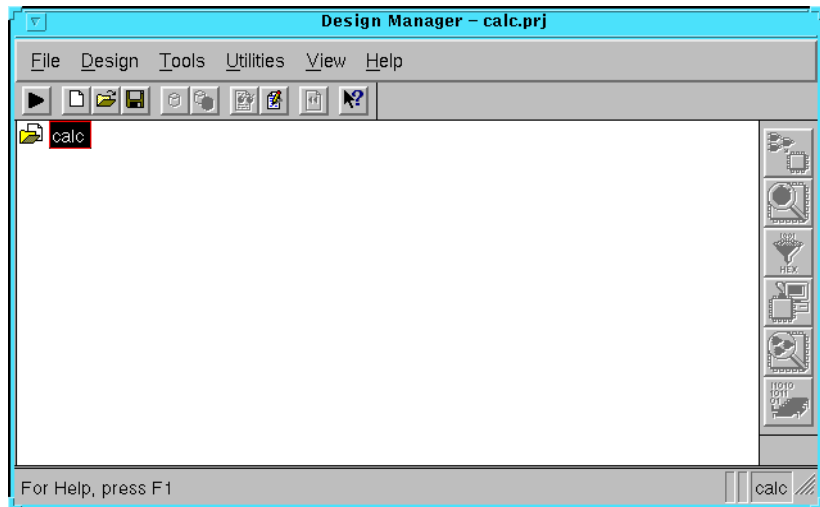


Figure 3-18 Xilinx Design Manager

Each project is associated with objects known as “versions” and “revisions.” Versions represent logic changes in a design (for example, adding a new block of logic, replacing an AND gate with an OR gate, or adding a flip-flop); revisions represent different executions of the design flow on a single design version, usually with new implementation options (for example, higher place and route effort, a change in part type, or experimentation with new bitstream options).

2. Within the Xilinx Design Manager, select **Design** → **Implement**.

The Implement dialog box opens as shown in the following figure and displays fields for part type, design version, and revision.

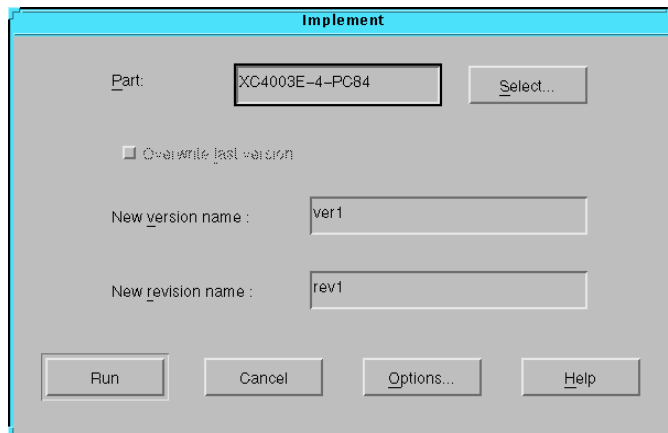


Figure 3-19 Implementation Dialog Box

3. The Xilinx Design Manager reads the part type from the design.

If you wish to specify the part type manually, click the **Select** button to display a pull-down listing of available devices. Choose a family, a device, a package, and a speed grade. Click **OK**. The part number is inserted into the **Part** field in the **Implement** dialog box.

4. Click on **Options**. The Options dialog box appears as shown in the **“Options Dialog Box”** figure.

Note: The CPLD Options dialog box does not have a Configuration Template section, nor does it have a Produce Logic Level Timing Report checkbox.

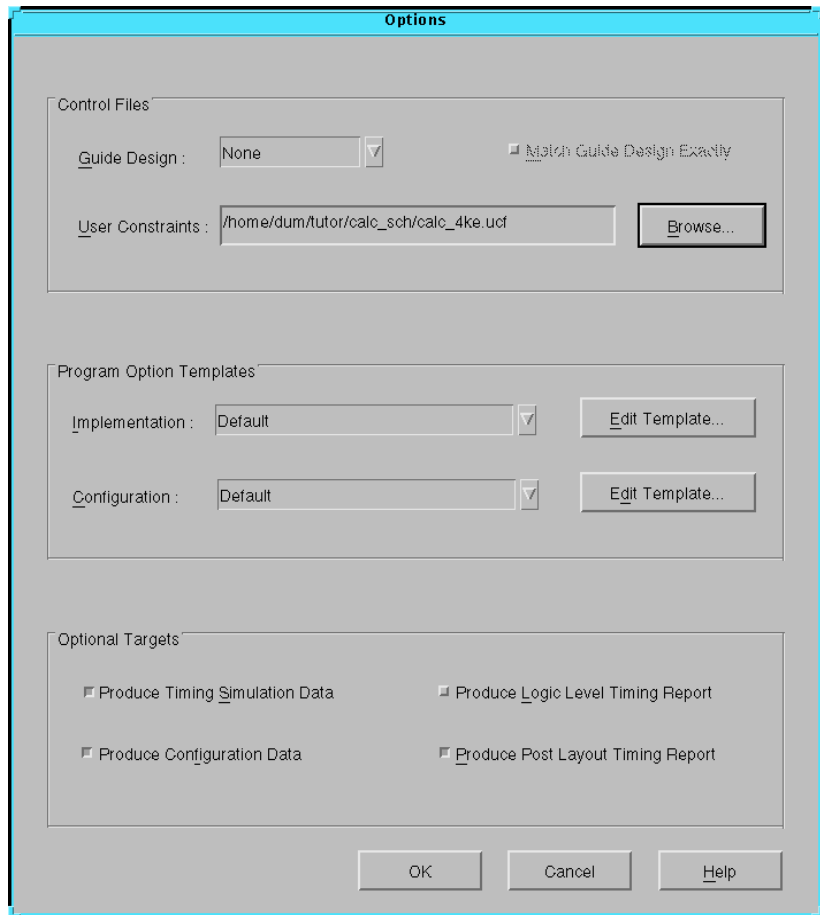


Figure 3-20 Options Dialog Box

5. Click **Browse** next to the User Constraints field. Select the appropriate .ucf file from the design directory, then Click **OK**.
6. Under Optional Targets, make sure the following are selected:
 - **Produce Timing Simulation Data:** This generates a back-annotated EDIF netlist that can be imported into the Mentor Graphics tools.
 - **Produce Configuration Data:** This generates a programming bitstream suitable for downloading into the Xilinx device.

- **Produce Post Layout Timing Report:** This generates a timing report file based on how the design is actually routed.

You can also select the following option (FPGAs only):

- **Produce Logic Level Timing Report:** This generates a preliminary (pre-place and route) timing report based on the number of logic levels in each signal path. Since it is generated before the place-and-route layout step, it does not contain information on device routing. Looking at this report before place and route can be useful for seeing how much “routing slack” you have in a design.

7. Select the **Edit Template** button on the right hand side of the Implementation field. The Implementation Options dialog box appears as shown in the following figure.

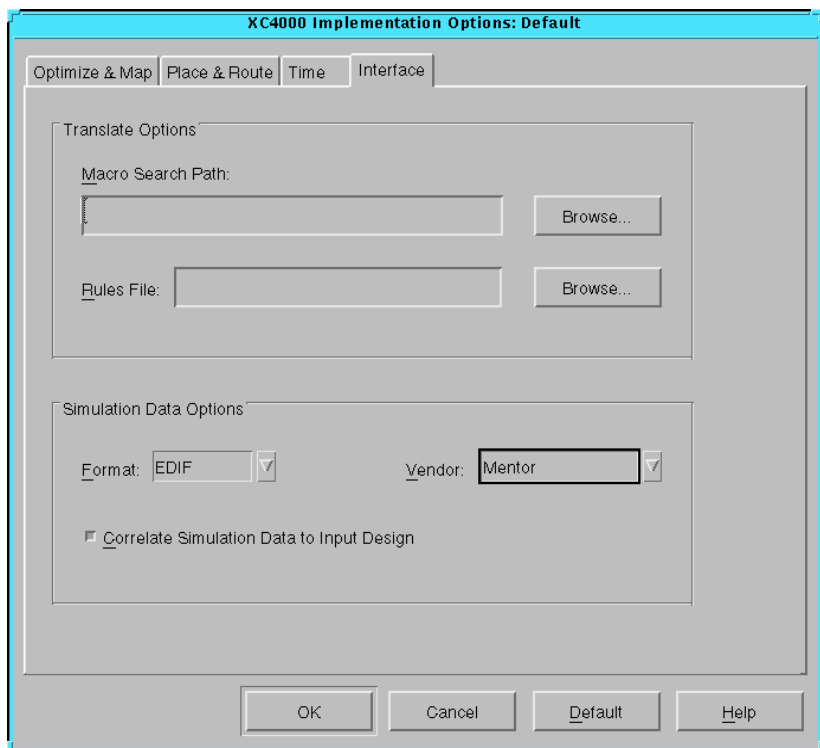


Figure 3-21 Changing EDIF Vendor Information

8. Select the **Interface** tab. In the Interface pane, look under Simulation Data Options and verify that Format is set to EDIF and that Correlate Simulation Data to Input Design is selected. In the Vendor field, select Mentor.
9. Click **OK** to return to the Options window.
10. Click **OK** to return to the Implementation dialog box.
11. In the Xilinx Design Manager window, verify that you have selected the current version and revision you wish to work on, then click **Run**. The Flow Engine comes up as shown in the following figure.

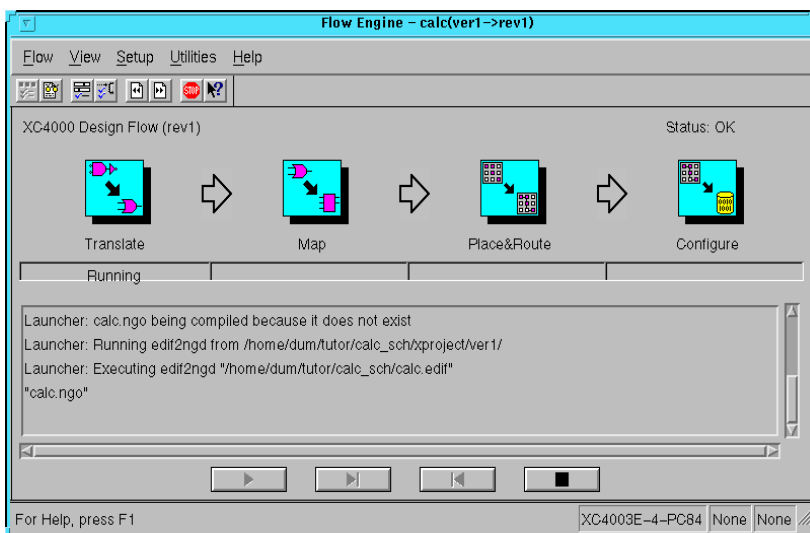


Figure 3-22 The Xilinx Flow Engine

The status bar shows the progress of the implementation flow with the following stages:

- **Translate:** Convert the design EDIF file into an NGD (Native Generic Design) file.
- **Map:** Group basic elements (bels) such as flip-flops and gates into logic blocks (comps). Also generate a logic-level timing report if desired.
- **Place&Route:** Place comps into the device, and route signals between them.

- **Timing:** Generate timing simulation data and an optional post-layout timing report.
 - **Configure:** generate a bitstream suitable for downloading into and configuring a device
12. When the implementation completes, an Implementation Status box appears with:

```
Implementing revision ver1->rev1 completed  
successfully.
```

Click on **View Logfile** to display the logfile from Flow Engine. The report is displayed in vi. To exit the viewer, type **:q!** and press Return. Click **OK** in the Implementation Status dialog to return to the Xilinx Design Manager.

Note: To use another text editor, such as Emacs, as the report viewer, select **File** → **Preferences** from the Xilinx Design Manager.

For schematic-based designs, the Xilinx Design Manager produces an EDN file, which is a post-route EDIF netlist file that expresses timing and simulation in SIMPRIM library elements instead of Unified Libraries elements. You can now submit the EDN file to `pld_edif2tim` to create a simulation model for timing simulation. This is described in the following section.

Timing Simulation for Schematic Designs

Timing simulation verifies design functionality by using delay information from the EDIF, VHDL, or Verilog file created during design implementation. It also describes how to perform a timing analysis with Mentor's QuickPath tool.

During design implementation, the Xilinx Design Manager can produce an EDIF (EDN) file. For EDIF files, the process of timing simulation consists of converting the EDIF netlist to a Mentor EDDM model, creating a component and a viewpoint, and simulating the design with `pld_quicksim`. The timing simulation process for EDIF files is shown in the **“Timing Simulation for Schematics”** figure.

The **“Performing Timing Simulation”** section of the **“Schematic Design Tutorial”** chapter illustrates the steps involved in timing simulation.

This section describes how to use QuickSim to perform timing simulation on designs described in EDIF.

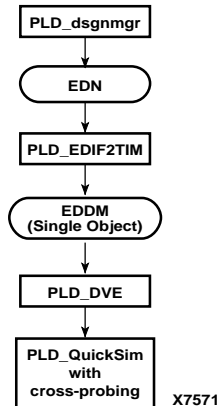


Figure 3-23 Timing Simulation for Schematics

Creating the EDDM Model and the Viewpoint

The first step in performing a timing simulation on your design is to use the `pld_edif2tim` utility to convert the EDIF netlist created by the Xilinx Design Manager to a Mentor EDDM model. At the same time, `pld_edif2tim` automatically creates a viewpoint which is subsequently processed by `pld_dve -s` to prepare it for timing simulation.

1. Double-click the left mouse button on the `pld_edif2tim` icon in the Design Manager Tools window.

The dialog box shown in the following figure appears.

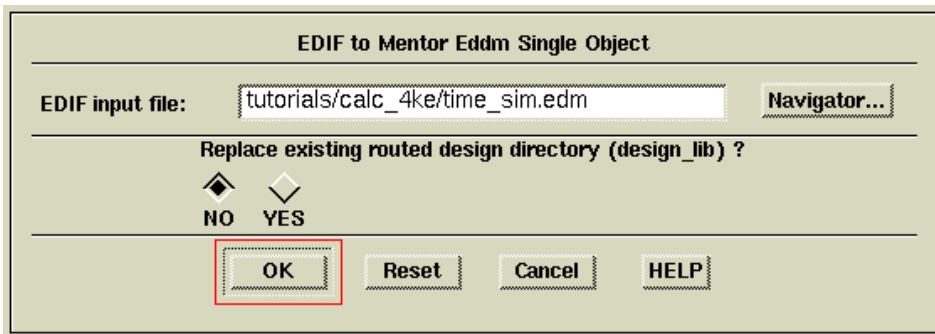


Figure 3-24 EDIF to Mentor Eddm Single Object Dialog Box

2. Enter the name of the EDN file in the EDIF Input File field, or click on **Navigator** to browse the available files.

The component created from the EDN file is put into a design library called *my_design_lib*. If you have already implemented your design at least once, this directory already exists. If you don't wish to copy over it, move it to another directory before proceeding.

3. Click on **OK**.
4. Invoke DVE, by double-clicking the left mouse button on the `pld_dve` icon in the Design Manager Tools window.

The dialog box shown in the following figure appears.

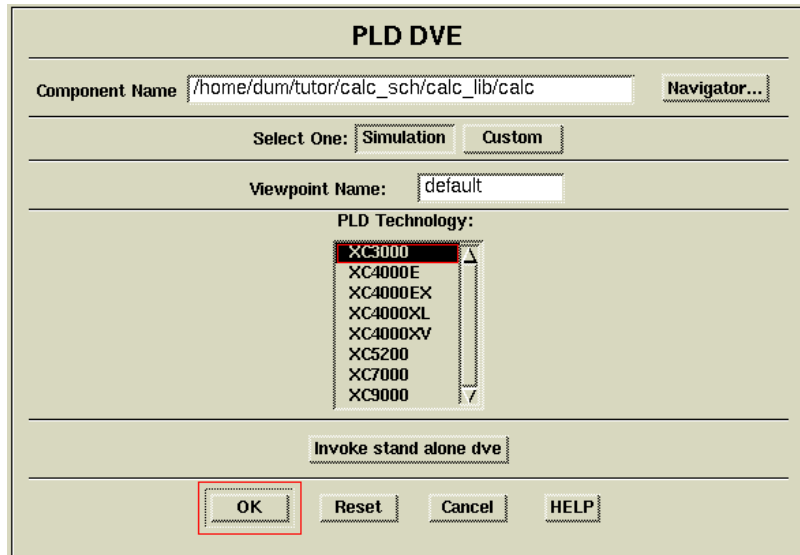


Figure 3-25 Pld_dve Dialog Box

5. Enter the top-level component name created by `pld_edif2tim` in the `my_design_lib` directory.
6. Use the **Navigator** button to navigate all the way down to the “default” viewpoint and select the viewpoint.
7. Select the **Simulation** Button.
8. Select the appropriate technology from the PLD Technology box.
9. Click **OK**.

Simulating the Design

You can now submit the design to `pld_quicksim` for timing simulation.

1. To invoke `pld_quicksim`, double-click the left mouse button on the `pld_quicksim` icon in the Design Manger Tools window.

The `pld_quicksim` 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.

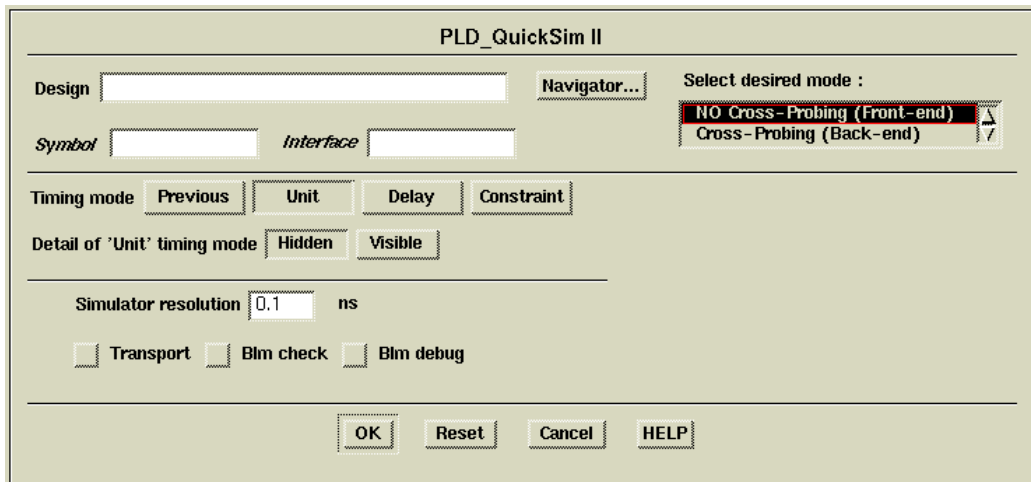


Figure 3-26 PLD_QuickSim II Dialog Box

2. In the Design field, enter the name of the top-level design created by pld_edif2tim.
3. In the Select Desired Mode field, select **Cross-Probing**.

Normally, you select cross-probing for back-end EDDM designs but not for front-end designs. You can only cross-probe back-end designs that contain either pure schematic or schematics that contain EDDM hierarchical models. You cannot cross-probe designs written in HDL or that contain HDL models. See the [“Cross-Probing” section](#) for more details about cross-probing.

Warning: In order for cross-probing to work, other sessions of Design Viewpoint Editor and QuickSim must be closed. Otherwise, the inter-process communication gets confused. This includes another user’s session invoked by rlogin from another workstation.

4. Set the timing modes as desired.
5. Click on **OK**.

Pld_quicksim now simulates the design. The QuickSim graphical user interface appears. If you selected cross-probing, DVE is invoked as well.

6. In DVE, open the viewpoint of the front-end schematic design, that is, the viewpoint submitted to pld_men2edif.

7. Open the sheet of the design, and select signals that you wish to trace.

These signals are automatically added to the QuickSim trace window. If you have a file that sets up your trace window and stimulus, use that instead. Any signals selected in the trace window select the same on the schematic on which they reside in the DVE window. If such sheets have not been opened in DVE yet, you must open them to see the selections.

Cross-Probing

Cross-probing is a way of cross-referencing between the original schematic and the timing simulation model after placement and routing. Once a Mentor design is translated, expanded, mapped, placed, and routed, you can extract the back-annotation information and create a hierarchical EDIF netlist. After you convert this EDIF to an EDDM model using `pld_edif2tim`, you submit it to `pld_dve` to create a viewpoint and then to `pld_quicksim` for timing simulation. The resulting data base preserves the design hierarchy, and although it is created in terms of the SIMPRIM library, most of the original net names are still available. You enable cross-probing by invoking QuickSim with the `-cp` option. This option invokes `pld_dve` as well as `pld_quicksim`. You then open the original design viewpoint in `pld_dve` and view the desired design sheet. If you display the original schematic in `pld_dve`, you can select nets on the original schematic and view them in the QuickSim trace window.

You may optionally create a schematic model using Mentor's schematic generator (`sg`) from the Eddm model created by `pld_edif2tim`. This schematic is only for viewing the backend schematic and is not required for the Xilinx flow to work. With cross-probing, you can use your original schematic for this purpose.

You should usually be able to reapply your original test vectors to the new `Eddm_single_object` design model for timing and/or functional simulation in QuickSim.

When you create the trace/list window in QuickSim, selecting signals from the original selected test vectors should cause the corresponding net on the original schematic sheet in `pld_dve` to be selected. If it is unselected in the trace/list window, it is also unselected on the original schematic.

If a net is selected in the pld_dve schematic sheet window, the net is automatically added to QuickSim trace window. If the net due to optimization or other complexities has been eliminated in the post-layout design, QuickSim issues an Error message of the type:

```
Error: $$add_traces returned error status at line 440  
of file /tools/...
```

```
Error: Unable to resolve string '/ALU/I$10/C2' to a  
signal or expression
```

No trace is displayed for this net.

When a net is selected on the original schematic sheet in pld_dve and if the corresponding signal is already added to the trace/list window, the net will not be added again; instead, it is highlighted in the trace/list window.

Adding list windows in quicksim is your choice. List windows are not automatically created. If you do create a list window, it is your choice which signals to add to the list window. Opening a list window does not automatically show or add the signals from the trace window. However once you have added signals to the list window, selecting such signals will interact with the original schematic exactly the same way as the ones in trace window.

Warning: In order for cross-probing to work, other sessions of Design Viewpoint Editor and QuickSim must be closed. Otherwise, the inter-process communication gets confused. This includes another user's session invoked by rlogin from another workstation to your workstation.

Note: If you flatten your design during netlist generation, you lose hierarchical aliases for signals that span multiple hierarchy levels; only the name of the signal at its highest level is preserved.

While 100% backannotation is possible, certain limitations of simulators, optimization process, and modelling of complex functions can make 100% back annotation impossible.

For more details on cross-probing, see the following sections:

- [“Performing Timing Simulation” section of the “Schematic Design Tutorial” chapter](#)
- [“Invoking Pld_quicksim” section of the “Schematic Design Tutorial” chapter](#).

Performing a Timing Analysis

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 on QuickPath.

You can perform a timing analysis either before or after timing simulation; however, you may want to perform the timing simulation first to assure the design's functionality, then use QuickPath to determine the design's critical path.

Note: Running QuickPath on PLD designs is optional.

1. To start QuickPath, double-click the left mouse button on the QuickPath icon in the Design Manager Tools 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.

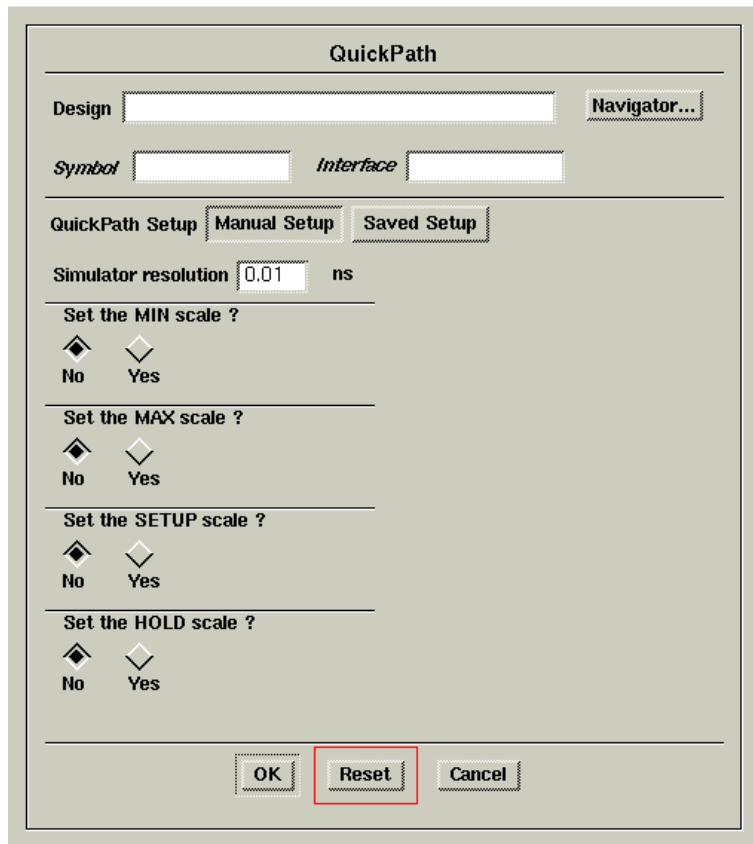


Figure 3-27 QuickPath Dialog Box

2. Enter the name of the design in the Design field, or click on **Navigator** to find the design.
3. Do not enter anything in the Symbol or Interface field unless the top-level design contains more than one interface. If the top-level design has more than one component interface table or symbol, you can specify the source by entering the name in the appropriate field.
4. In the QuickPath Setup field, select **Manual** unless you have saved a previous timing analysis environment and want to load that file that contains it.

5. In the Simulator Resolution box, enter the smallest unit of time that you want to be visible in the simulator.

The smallest resolution allowed for Xilinx designs is 0.1 ns.

6. In the Set the MIN Scale, Set the MAX Scale, Set the SETUP Scale, and Set the HOLD Scale fields, click on **No** unless you want to perform a “corner” analysis.

These fields set scaling values that govern the minimum propagation delay, the maximum propagation delay, the setup time, and the hold time, respectively. In a “corner” analysis, you use these scaling values to perform two timing analyses. The first analysis models a slow chip and the second models a fast chip. If a chip with values at both extremes of the timing spectrum successfully passes the timing analysis, it is likely that all chips with values in between will also pass. These scaling values allow you to look at the differences in timing due to the variations in device process, voltage, and temperature.

7. Click on **OK** to start the timing analysis.

A session window with a menu bar, messages window, and Setup/Analysis palette now opens.

Click on **Open Sheet** to display the top-level design, and then proceed with the timing analysis.

HDL Designs

This chapter describes how to use the Mentor Graphics Interface to design with pure HDL designs. It contains the following sections:

- “The Design Flow” section
- “HDL Design Entry” section
- “Functional Simulation” section
- “Design Implementation” section
- “Timing Simulation” section

The Design Flow

The following figure shows the design flow for VHDL and Verilog design entry, functional simulation, synthesis, and timing simulation for all supported technologies.

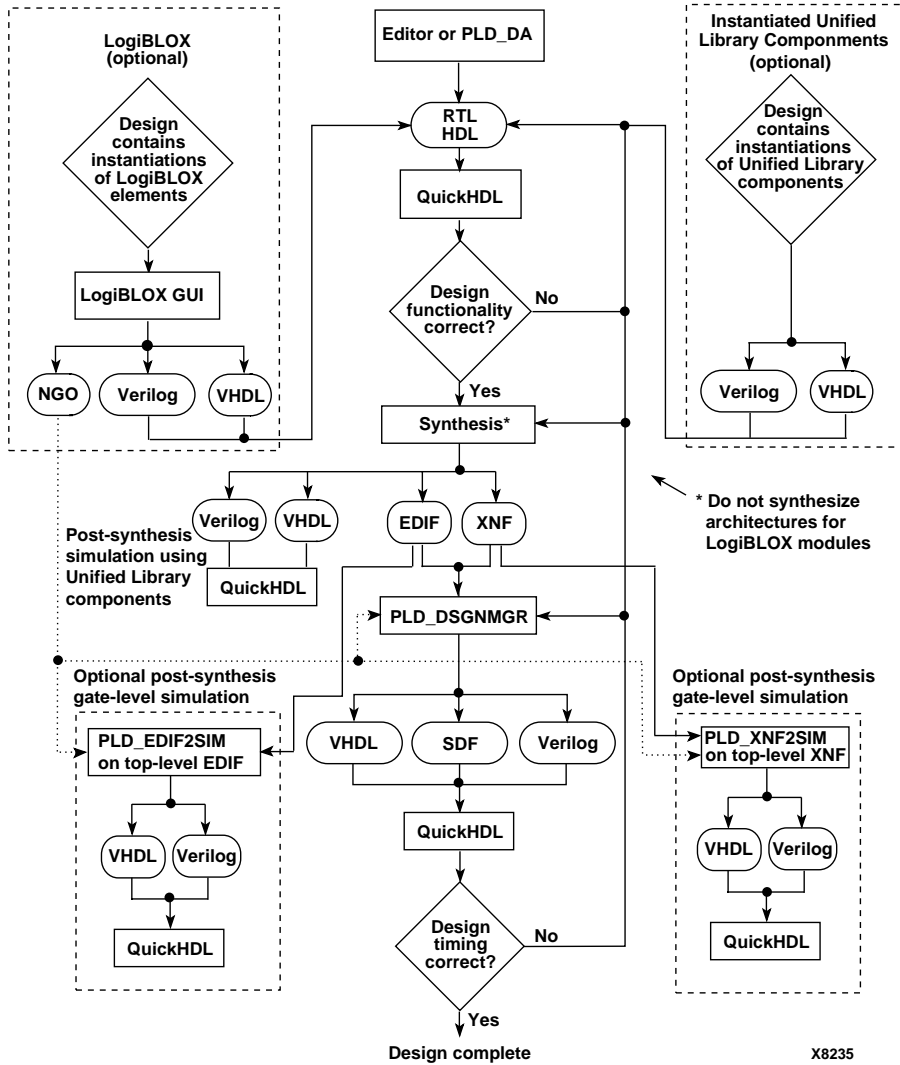


Figure 4-1 HDL (Verilog/VHDL) Design Flow

HDL Design Entry

This section describes the basic process of entering HDL designs. In addition to this chapter, the HDL design entry techniques in this section apply to the “Mixed Designs with VHDL on Top” chapter and “Mixed Designs with Schematic on Top” chapter.

Overview of HDL Design Entry

Use a text editor, `p1d_da`, System Architect, Renior, or other HDL entry tool that is compatible with your synthesizer to create synthesizable VHDL or Verilog. `P1d_da` can be better than a plain text editor for editing your source. With `p1d_da` you can submit the source to be compiled as you edit it (see the mentor *Design Architect User's Guide* for details). When performing HDL design entry, observe the following requirements:

- The synthesizers must create EDIF or XNF that is compatible with Xilinx implementation tools.
- Xilinx-specific properties and timing constraints cannot be added in a VHDL or Verilog description, but synthesizers do have the capability of adding them to the output EDIF or XNF via constraint setting options. These constraint settings must be consistent with the current Xilinx implementation tools requirements. Otherwise, implementation can be controlled within the implementation tools themselves or with a UCF (user constraint file) file.

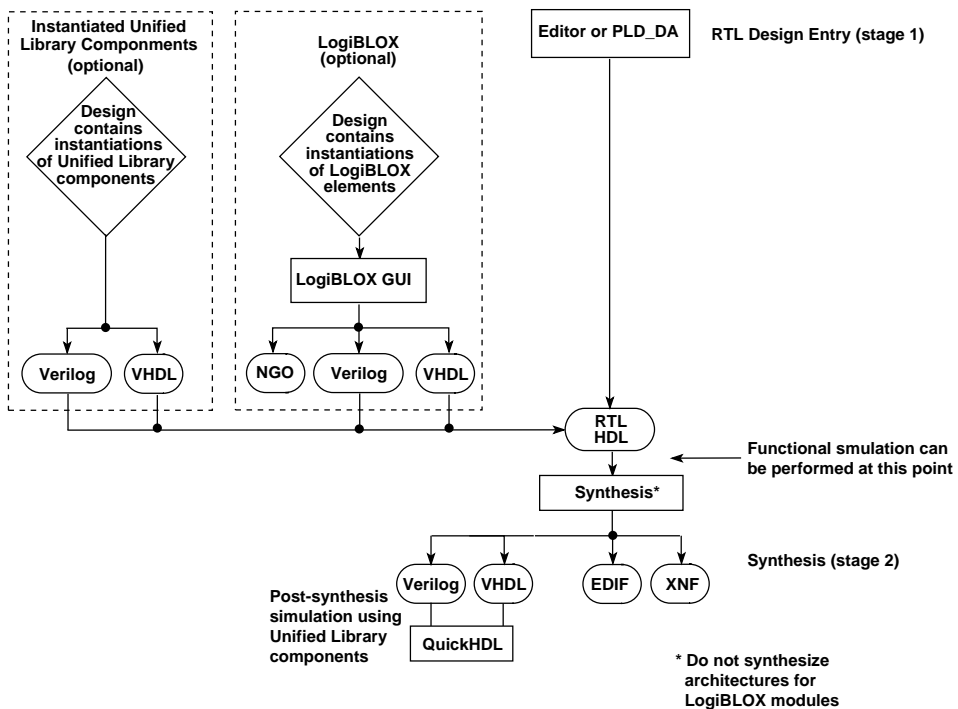
As an optional part of the Xilinx HDL design entry flow, you can instantiate LogiBLOX modules in your VHDL or Verilog designs and simulate the HDL output from LogiBLOX in your HDL simulators. When using LogiBLOX modules in HDL design entry, observe the following requirements:

- Create the NGO from LogiBLOX for later use in the implementation tool. LogiBLOX NGO files must be placed in your top level directory or you must modify the macro search path in the Xilinx Design Manager to include the location of NGO files. `P1d_edif2sim` or `p1d_xnf2sim` do not have the macro search path functionality. You must have the LogiBLOX NGO files in the same directory as your top-level EDIF or XNF.

- Your synthesizer must not read-in or synthesize the HDL description of the LogiBLOX modules. These descriptions are for simulation only. The modules must be treated as black boxes by the synthesizer.

HDL Design Entry Stages

HDL design entry has two stages as shown in “HDL (Verilog/VHDL) Design Entry and Synthesis” figure. The first stage is the Register Transfer Level (RTL). At this level, the design behavior is described in a high-level, non-technology-specific manner. Instantiation of specific components is the exception. An example would be RAMs or LogiBLOX modules. This design entry step is generally followed by a functional simulation.



x8233

Figure 4-2 HDL (Verilog/VHDL) Design Entry and Synthesis

During design entry, you may check out the syntax correctness of your code by compiling it for your synthesizer and/or QuickHDL without doing either the synthesis or simulation.

You may find the syntax checkers are different. The synthesizer may check for certain constructs it cannot synthesize like textio in VHDL, but these constructs may be perfectly good for simulating functionality as you develop the circuit. Many synthesizers have pragma or meta comments that allow you to keep this code in your HDL description but tell the synthesizer to ignore it.

Also there may be significant differences in how thoroughly the compilers check the code against the VHDL and Verilog IEEE-standards or even how they interpret certain sections.

It is good practice to do occasional compiles for both the synthesis tool and QuickHDL as you develop large sections of your HDL code.

Once the RTL simulation is correct, the second stage of design entry is to submit the RTL code to a synthesizer where the general functionality is synthesized and mapped to gates in a specific technology. At this point you have the option of performing a second functional simulation of the post-synthesis gate level description. However, this step is not necessary since no additional timing information is available before place and route.

Once you are satisfied with the behavior of the circuit, you can send the gate level output of the synthesizer to the implementation tool as either an XNF or EDIF file.

Stage 1: RTL Behavioral Code Development

The first stage of HDL design entry is developing an RTL behavioral description. Code created at the RTL entry is generally non-technology specific. Two exceptions worth noting are:

- Coding style favoring one technology strengths over another.
- Instantiating technology specific components.

Your coding style should take into account your targeted technology's architecture specifics to achieve the best performance or smallest size for your design.

For example, a style that infers latches unintentionally or even on purpose would be just fine for a XC5200 or XC4000EX part but would be a trouble spot with an XC3000. It is a problem for the XC3000

because it would be implemented as cross-coupled logic creating a host of timing analysis issues. In a XC4000 it would take up valuable resources since each latch could be implemented as a SRAM cell, taking up a whole function generator. Synthesizers may vary in how they implement latches in technologies that do not have explicit latches. Some may use cross-coupled logic as in the XC4000E. Others may use a RAM cell.

Another RTL coding style problem might be describing the functionality in a manner that infers lots of MUXES. A better choice in some technologies would be to infer internal tri-states and use the high-performance tri-state lines.

Some synthesizers may not have a means of inferring or targeting high performance technology components like wide-edge decoders, I/O muxes, or latches. In that case you may need to instantiate these components to get your best chip performance.

In summary, observe the following RTL coding guidelines:

- Avoid using latches in technologies without specific architecture components to support them like the XC5200 and XC4000EX have.
- Infer tri-state buses instead of muxes for technologies with good internal tri-state structures.
- Instantiate high performance architecture features if your synthesizer cannot infer them.

Stage 2: Synthesis

The second stage of HDL design entry is synthesis of the RTL behavioral description down to technology specific gates. Specific synthesis design entry steps for Xilinx parts are highly dependent on which synthesizer you use. Generally you can break synthesis design entry into the following steps:

1. Tailor your RTL code for both the synthesizer and the Xilinx technology's capabilities. For example, if your synthesizer can insert the STARTUP component, you need not instantiate it.

When you simulate in QuickHDL with an instantiated startup block, you get a warning because the startup module does not have a simulation module. You may ignore this warning since the startup block is only used to direct the implementation tool and

does not change the logical functionality of the circuit. The warning looks something like this:

```
# ** Warning: Component startupblk is not bound.
```

2. Guide the synthesis process with timing and/or size requirements.
3. Guide the output process to select XNF or EDIF outputs to insert I/O buffers, global buffers, STARTUP blocks, and to output timing constraints either within the netlist itself or to a separate file readable by Xilinx implementation tools. Make sure the timing constraint style is the correct one for the current version of Xilinx implementation tools.

You should read your synthesizer manual for specific details, especially the sections about targeting Xilinx devices. Be aware that any one of these three steps can greatly affect the quality of synthesis and/or implementation results.

LogiBLOX Design Entry

Some synthesizers are not capable of using Xilinx carry chains properly and tend to infer inefficient structures for adders, counters, etc. Other synthesizers are able to infer incrementors or decrementors, but do not use efficient logic for the control logic of the loadable counter.

To guarantee optimal synthesis for certain modules in a Xilinx technology, you may use the LogiBLOX module generator and instantiate the resulting module in your HDL code.

You may invoke the stand-alone Graphic User Interface of LogiBLOX in the tool window of `pld_dmgr` by clicking on the `pld_logiblox` icon. This is not the same Graphic User Interface you get in the Design Architect Schematic Palette. The stand-alone version is for VHDL or Verilog models only. Another way to invoke the stand-alone GUI is from the Design Architect pop-up menu in the Session Window.

LogiBLOX requires two outputs for proper implementation in a HDL design.

The first output is the HDL behavior description for simulating either VHDL or Verilog. These HDL descriptions only support HDL functional simulations. You should not send them to the synthesizer for synthesis. The entities can be used for component instantiation

purposes, but the architectures should be treated as black-boxes within the synthesizer. The following is an example of a LogiBLOX component declaration and instantiation in VHDL:

```
-----  
-- Component Declaration  
-----  
component RAM16X1  
  PORT(  
    A: IN std_logic_vector(3 DOWNT0 0);  
    DI: IN std_logic_vector(15 DOWNT0 0);  
    WR_EN: IN std_logic;  
    DO: OUT std_logic_vector(15 DOWNT0 0));  
end component;  
  
-----  
-- Component Instantiation  
-----  
instance_name : RAM16X1 port map  
  (A => ,  
   DI => ,  
   WR_EN => ,  
   DO => );
```

The second output is the NGO file. The implementation tools use this file to pull the LogiBLOX module into the top-level design. Since these NGO files are technology specific, you should generate a new NGO file each time you select a new Xilinx architecture. The HDL behavioral descriptions do not change.

Unified Library Instantiated Components

If you prefer, you may instantiate Unified Library components into the RTL design. The components you use should be primitives supported in the Xilinx family being targeted and also present in the synthesis tool's target library. For more information on Unified Library components, see the *Development System User Guide*.

Functional Simulation

Pure HDL designs consist of a RTL VHDL or Verilog model. You can optionally convert the synthesis output netlist to a gate-level HDL model and functionally simulate it. The flow diagram for performing

functional simulation on pure HDL designs is shown in the following figure.

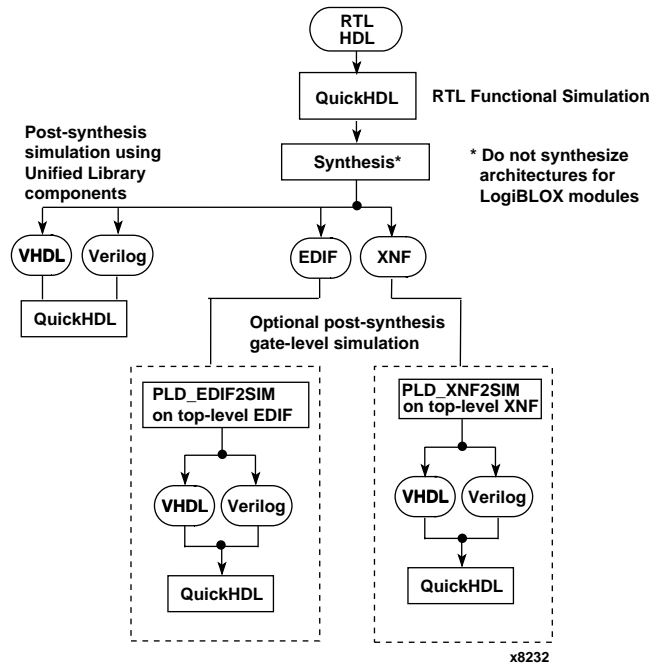


Figure 4-3 Performing Functional Simulation on a Pure HDL Design

Pre-Synthesis Functional Simulation

To perform a pre-synthesis functional simulation on a pure HDL design follow these steps:

1. Create a working library with qhlib.


```
qhlib work
```
2. Map the library with qhmap.


```
qhmap work mywork
```
3. If you are using LogiBLOX modules, use qhmap to map to the compiled LogiBLOX modules location.

`qhmap logiblox compiled_logiblox_area`

Your system administrator can tell you the location of the compiled version(s) of the LogiBLOX library. Instructions for compiling are in the Mentor Graphics Installation section of the *Alliance Installation Guide*. You may have to recompile the library for each version of QuickHDL you use. The default directory for the compiled LogiBLOX library is as follows:

```
$XILINX/mentor/data/vhdl/logiblox
```

4. If you are using Unified Library components, use `qhmap` to map to the compiled Unified Library location by executing the appropriate line below for the device family that you are using.

For vhdl:

```
qhmap unisim $XILINX/mentor/data/vhdl/unisim
```

```
qhmap unisim_5k $XILINX/mentor/data/vhdl/unisim_5k
```

Map to `unisim` for the XC3000 and XC4000 series, or `unisim_5k` for the XC5200 series.

For verilog:

```
qhmap uni3000 $XILINX/mentor/data/verilog/uni3000
```

```
qhmap uni4000x $XILINX/mentor/data/verilog/uni4000x
```

```
qhmap uni5200 $XILINX/mentor/data/verilog/uni5200
```

Map to `uni3000` for the XC3000 series, `uni4000x` for the XC4000 series, or `uni5200` for the XC5200 series.

Note: The above locations for the compiled libraries are the default locations for a default software installation. However, your system administrator can install them in other locations. Your system administrator can tell you the location of the compiled version(s) of the Unified Library. Instructions for compiling are in the Mentor Graphics Installation section of the *Alliance Installation Guide*. You may have to recompile the library for each version of QuickHDL you use.

5. Compile the HDL source files with `qvhcom` (VHDL) or `qvlcom` (Verilog).

```
qvhcom [options] design_file(s)
```

```
qvlcom [options] design_file(s)
```


See the Mentor documentation for a description of the available options.

6. Compile your testbench with qvhcom (VHDL) or qvlcom (Verilog).

```
qvhcom [options] testbench_file(s)
```

```
qvlcom [options] testfixture_file(s)
```

7. Select the appropriate architecture configuration or module for your testbench and select QHDL in the pld_dmgr tools window.

See the Mentor documentation for QuickHDL instructions.

8. After the RTL level simulation is correct, you may proceed to synthesis and to implementation or optional post-synthesis functional simulation.

Post-Synthesis Functional Simulation

To perform a post-synthesis functional simulation on a pure HDL design follow these steps:

1. Create a working library with qhlib.

```
qhlib work
```

2. Map the library with qhmap.

```
qhmap work mywork
```

3. If you are using LogiBLOX modules, use qhmap to map to the compiled LogiBLOX modules location.

```
qhmap logiblox compiled_logiblox_area
```

Your system administrator can tell you the location of the compiled version(s) of the LogiBLOX library. Instructions for compiling are in the Mentor Graphics Installation section of the *Alliance Installation Guide*. You may have to recompile the library for each version of QuickHDL you use. The default directory for the compiled LogiBLOX library is as follows:

```
$XILINX/mentor/data/vhdl/logiblox
```

4. Since you are using Unified Library components, use qhmap to map to the compiled Unified Library location by executing the appropriate line below for the device family that you are using.

For vhd1:

```
qhmap unisim $XILINX/mentor/data/vhdl/unisim
qhmap unisim_5k $XILINX/mentor/data/vhdl/unisim_5k
```

Map to unisim for the XC3000 and XC4000 series, or unisim_5k for the XC5200 series.

For verilog:

```
qhmap uni3000 $XILINX/mentor/data/verilog/uni3000
qhmap uni4000x $XILINX/mentor/data/verilog/uni4000x
qhmap uni5200 $XILINX/mentor/data/verilog/uni5200
```

Map to uni3000 for the XC3000 series, uni4000x for the XC4000 series, or uni5200 for the XC5200 series.

Note: The above locations for the compiled libraries are the default locations for a default software installation. However, your system administrator can install them in other locations. Your system administrator can tell you the location of the compiled version(s) of the Unified Library. Instructions for compiling are in the Mentor Graphics Installation section of the *Alliance Installation Guide*. You may have to recompile the library for each version of QuickHDL you use.

5. Compile the HDL source files with qvhcom (VHDL) or qvlcom (Verilog).

```
qvhcom [options] design_file(s)
qvlcom [options] design_file(s)
```

See the Mentor documentation for a description of the available options.

6. Compile your testbench with qvhcom (VHDL) or qvlcom (Verilog).

```
qvhcom [options] testbench_file(s)
qvlcom [options] testfixture_file(s)
```

7. Select the appropriate architecture configuration or module for your testbench and select QHDL in the pld_dmgr tools window.

See the Mentor documentation for QuickHDL instructions.

- After the post-synthesis simulation is correct, you may proceed to implementation.

Optional Post Synthesis Functional Simulation

You can optionally perform a post-synthesis functional simulation on a pure HDL design, by following these steps:

- Run `pld_edif2sim` on your top-level EDIF or `pld_xnf2sim` on your top level XNF file from synthesis.
- Specify either VHDL or Verilog output in the `pld_edif2sim` or `pld_xnf2sim` dialog box.
- Choose the Flat or Hierarchical option and click OK to create the structural HDL netlist.

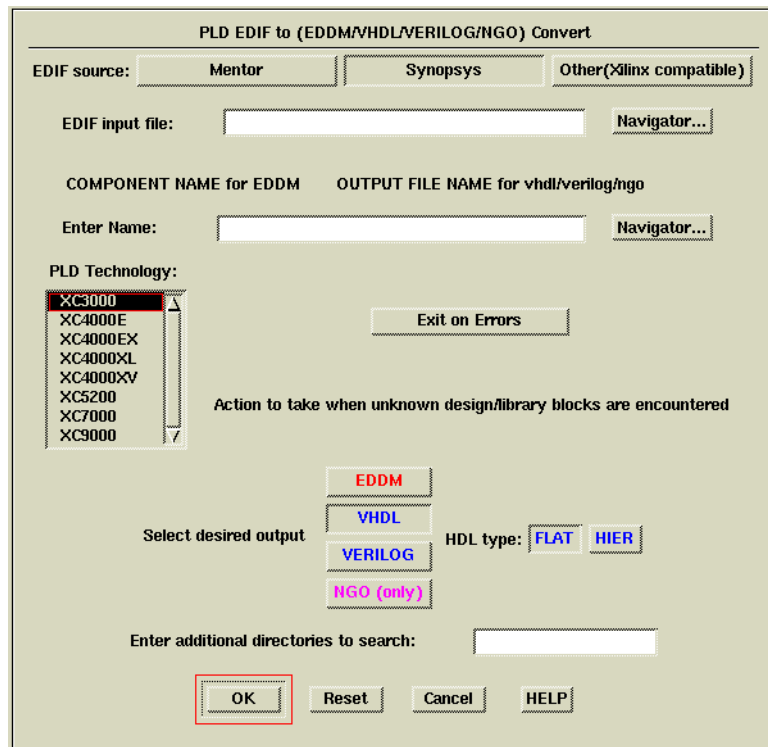


Figure 4-4 Pld_edif2sim Dialog Box

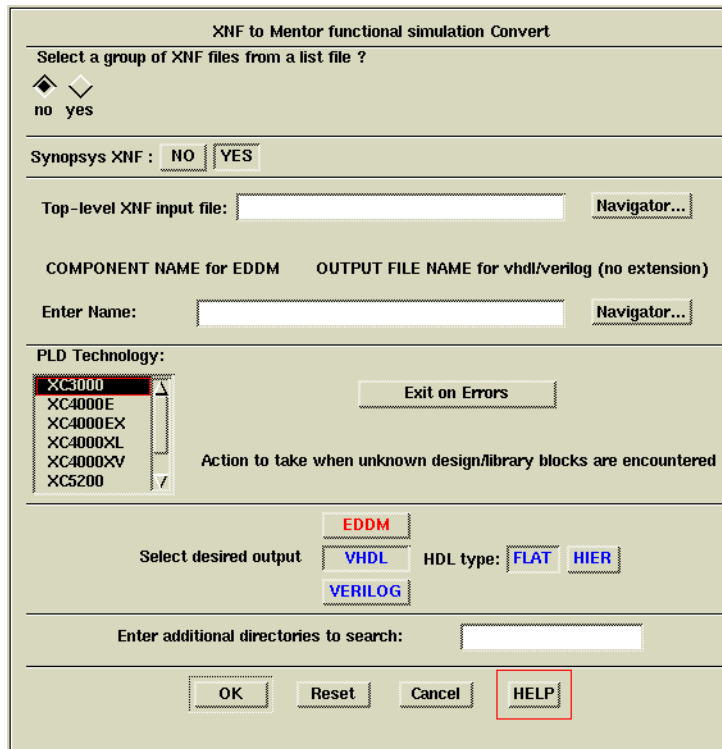


Figure 4-5 Pld_xnf2sim Dialog Box

4. Compile the HDL source file from `pld_edif2sim` or `pld_xnf2sim`.

Note: Before compiling, if you have not already done so, verify that the VHDL or Verilog SIMPRIM libraries have been compiled. Before performing timing simulation on an HDL-based design, the VHDL or Verilog SIMPRIM libraries must be compiled with `qvhcom/qvlcom`. Your system administrator should have performed this during installation.

The path to the VHDL libraries should be:

```
$XILINX/mentor/data/vhdl/simprim
```

The path to the Verilog libraries should be:

```
$XILINX/mentor/data/verilog/simprim
```

If these compiled SIMPRIM Libraries do not exist, contact your systems administrator. The Mentor Graphics Installation section

of the *Alliance Installation Guide* describes how to compile the SIMPRIM libraries.

5. Use qhmap to add a SIMPRIM library listing in the quickhdl.ini file:

```
qhmap simprim compiled_simprim_library_directory
```

The locations of the compiled simprim libraries (*compiled_simprim_library_directory*) are normally as follows:

```
$XILINX/mentor/data/vhdl/simprim
```

```
$XILINX/mentor/data/verilog/simprim
```

6. Type the following on the UNIX command line:

```
qvhcom [options] design_name (VHDL)
```

```
qvlcom [options] design_name (Verilog)
```

See the Mentor documentation for information on how to use qvhcom and qvlcom.

7. Select the appropriate architecture configuration or module for your testbench.
8. After the post-synthesis simulation is correct, you may proceed to implementation.

Design Implementation

Once you complete functional simulation for HDL designs, you are ready to implement your design in an FPGA or CPLD. You perform implementation with the Xilinx Design Manager, a graphical design-flow and project manager. In the Mentor interface, the Xilinx Design Manager is called pld_dsgnmgr. You invoke pld_dsgnmgr from the Mentor Design Manager or from a UNIX shell.

The “**HDL Design Implementation**” figure shows the design flow for implementing a design. The Xilinx Design Manager accepts your design, represented by the XNF or EDIF file from the synthesis tool. Design entry of pure HDL designs, or HDL designs with LogiBLOX elements produces an EDIF or XNF file that you can submit to pld_dsgnmgr. Pld_dsgnmgr first translates the design into a flattened or hierarchical netlist, then optimizes, places, and routes the design. You can also use the Xilinx Design Manager to generate SDF timing information that you can import into QuickHDL along with your

VHDL or Verilog netlist. For a more in-depth discussion of the flow, including advanced implementation options, see the *Development System User Guide*.

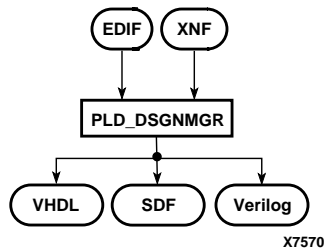


Figure 4-6 HDL Design Implementation

To implement your design follow these steps:

1. Within the Mentor Design Manager, select the EDIF icon for your design in the Navigator, then select **Right Mouse Button** → **Open** → **p1d_dsgnmgr**. The Xilinx Design Manager appears as shown in the “**Xilinx Design Manager**” figure. The tool automatically creates a Xilinx project called *your_design_name*. Xilinx project information is kept in a file called `xproj/your_design_name.prj` by default.

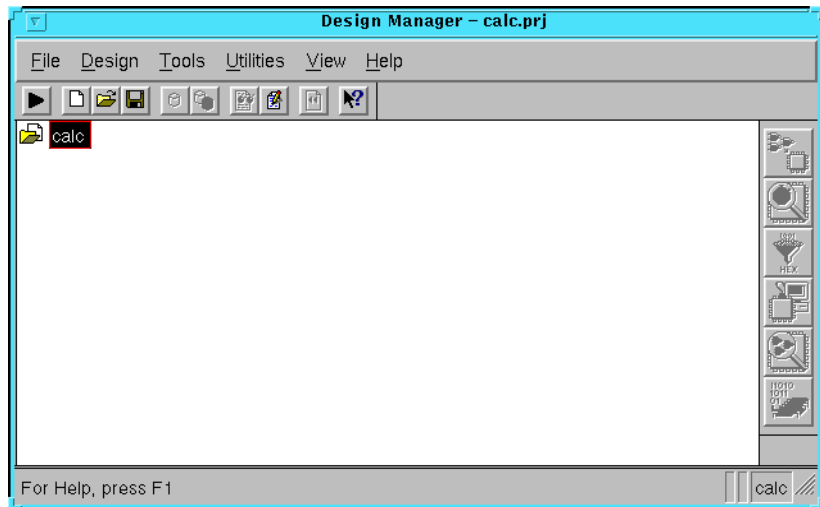


Figure 4-7 Xilinx Design Manager

Each project is associated with objects known as *versions* and *revisions*. Versions represent logic changes in a design (for example, adding a new block of logic, replacing an AND gate with an OR gate, or adding a flip-flop); revisions represent different executions of the design flow on a single design version, usually with new implementation options (for example, higher place and route effort, a change in part type, or experimentation with new bitstream options).

2. Within the Xilinx Design Manager, select **Design** → **Implement**.

The Implement dialog box opens as shown in the following figure and displays fields for part type, design version, and revision.

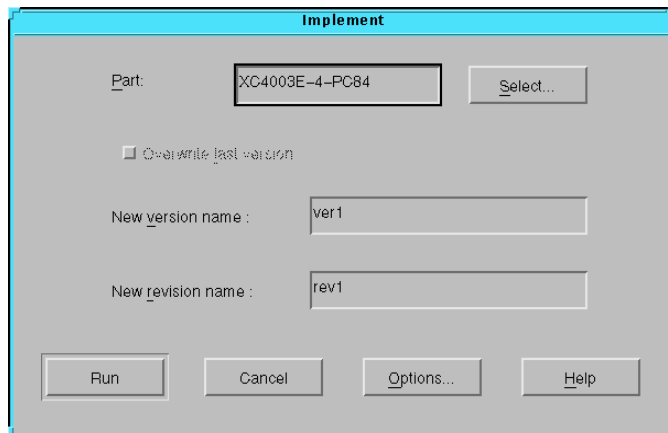


Figure 4-8 Implement Dialog Box

3. The Xilinx Design Manager reads the part type from the design.

If you wish to specify the part type manually, click the Select button to display a pull-down listing of available devices. Choose a family, a device, a package, and a speed grade. Click OK. The part number is inserted into the Part field in the Implement dialog box.

4. Click on Options. The Options dialog box appears as shown in the **“Options Dialog Box”** figure.

Note: The CPLD Options dialog box does not have a Configuration Template section, nor does it have a Produce Logic Level Timing Report checkbox.

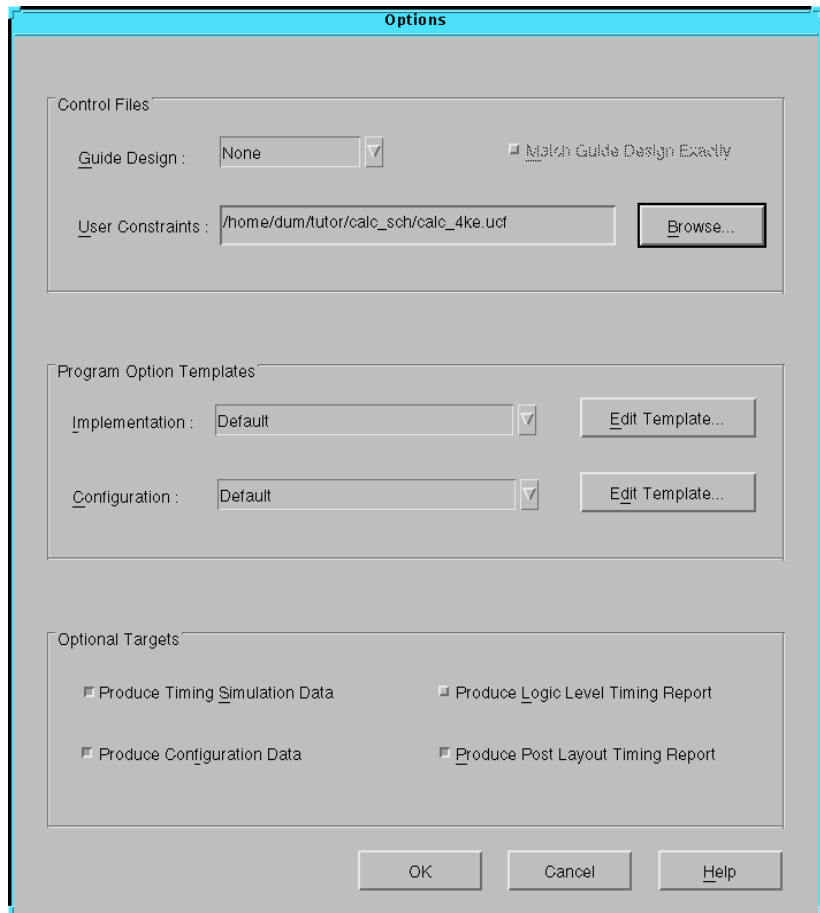


Figure 4-9 Options Dialog Box

5. Click the **Browse** button next to the User Constraints field. Select the appropriate .ucf file from the design directory, then Click **OK**.
6. Under Optional Targets, make sure the following are selected:
 - **Produce Timing Simulation Data:** This generates a SDF file and a VHDL or Verilog netlist that you can import into QuickHDL.
 - **Produce Configuration Data:** This generates a programming bitstream suitable for downloading into the Xilinx device.

- **Produce Post Layout Timing Report:** This generates a timing report file based on how the design is actually routed.

You can also select the following option (FPGAs only):

- **Produce Logic Level Timing Report:** This generates a preliminary (pre-place and route) timing report based on the number of logic levels in each signal path. Since it is generated before the place-and-route layout step, it does not contain information on device routing. Looking at this report before place and route can be useful for seeing how much routing slack you have in a design.

7. Select the **Edit Template** button on the right hand side of the Implementation field. The Implementation Options dialog box appears as shown in the following figure.

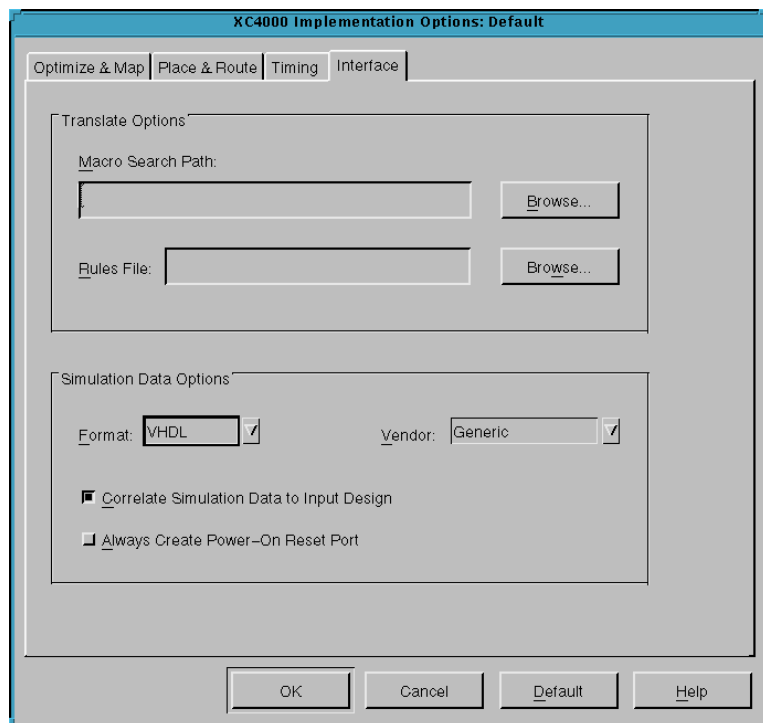


Figure 4-10 Changing Simulation Data Option

8. Select the **Interface** tab. In the Interface pane, look under Simulation Data Options and verify that Format is set to VHDL or Verilog and that Correlate Simulation Data to Input Design is selected. The Vendor field should change to generic since these HDL outputs are not vendor specific.
9. Click **OK** to return to the Options window.
10. Click **OK** to return to the Implementation dialog box.
11. At this point you have the option to create a template testbench for the timing VHDL or Verilog netlist. To do so, perform these steps:
 - a) In the Xilinx Design Manager, select **Utilities** → **Template Manager** to open the Template Manager dialog box.

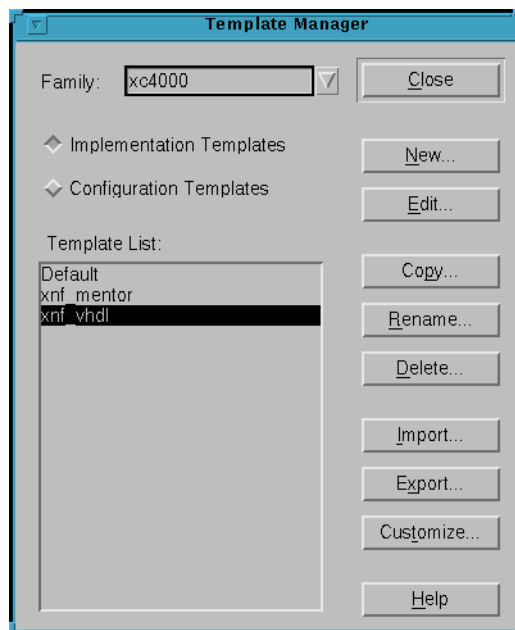


Figure 4-11 Template Manager Dialog Box

- b) Select the implementation template that you wish to customize.

- c) Click on the Customize button to open the Customize dialog box.

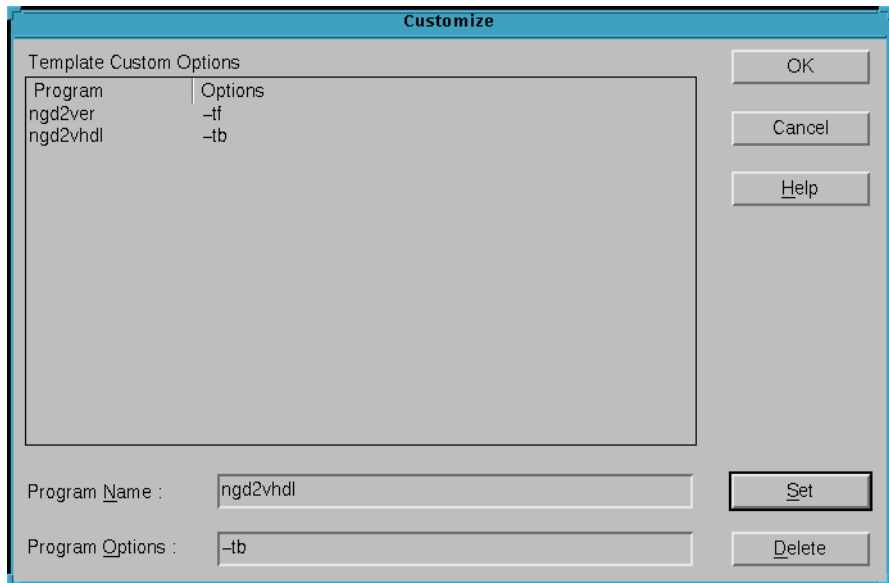


Figure 4-12 Customize Dialog Box

- d) If you have Verilog, specify ngd2ver in the Program Name box and -tf in the Program Options box.
- e) If you have VHDL, specify ngd2vhdl in the Program Name box and -tb in the Program Options box.
- f) Click Set and then OK.
- g) Click Close in the Template Manager dialog box.

Whenever you output VHDL or Verilog with this customized implementation template, the testbench template file *your_design.tvhd* is created for VHDL designs and *your_design.tv* is created for Verilog designs.

- 12. The default hierarchy for VHDL or Verilog output files is flat since the write function and simulations are generally faster. At this point you have the option to retain the hierarchy to aid debugging. To do so, follow these steps:

- a) In the Xilinx Design Manager, select **Utilities** → **Template Manager** to open the Template Manager dialog box as shown in “**Template Manager Dialog Box**” figure.
- b) Select the implementation template that you wish to customize.
- c) Click on the Customize button to open the Customize dialog box as shown in the “**Customize Dialog Box**” figure.
- d) If you have Verilog, specify ngd2ver in the Program Name box and -r in the Program Options box.
- e) If you have VHDL, specify ngd2vhdl in the Program Name box and -r in the Program Options box.
- f) Click Set and then OK.
- g) Click Close in the Template Manager dialog box.

Whenever you output VHDL or Verilog with this customized implementation template, the testbench template file *your_design.vhd* is hierarchical for VHDL designs and *your_design.v* is hierarchical for Verilog designs.

13. In the Xilinx Design Manager Implement dialog box, specify the current version and revision you wish to work on, then click Run.

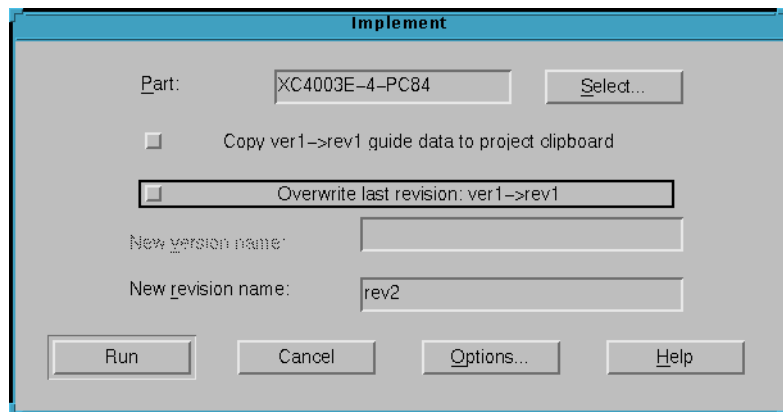


Figure 4-13 Implement Dialog Box

The Flow Engine opens as shown in the following figure.

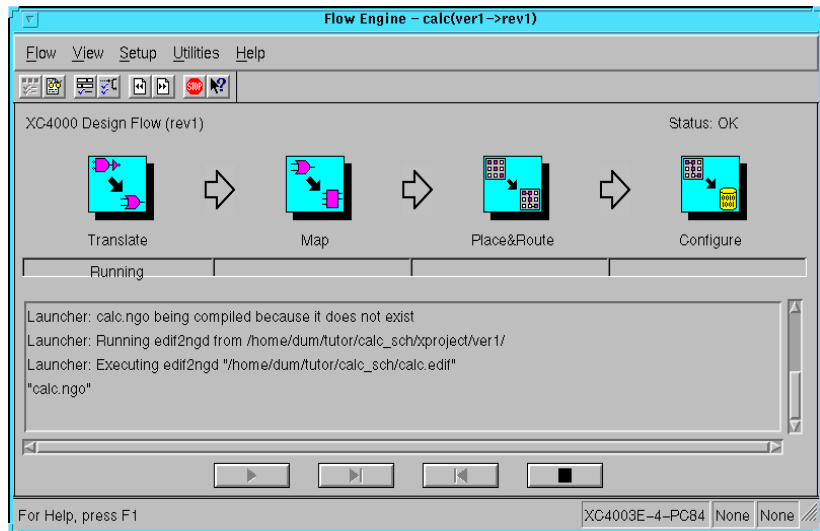


Figure 4-14 The Xilinx Flow Engine

The status bar shows the progress of the implementation flow with the following stages:

- **Translate:** Converts the design EDIF or XNF file into an NGD (Native Generic Design) file.
- **Map:** Groups basic elements (bels) such as flip-flops and gates into logic blocks (comps). Also generates a logic-level timing report if desired.
- **Place&Route:** Places comps into the device, and routes signals between them.
- **Timing:** Generates timing simulation data and an optional post-layout timing report.
- **Configure:** generates a bitstream suitable for downloading into and configuring a device

When the implementation completes, an Implementation Status box appears with:

```
Implementing revision ver1->rev1 completed
successfully.
```

14. Click on View Logfile to display the logfile from Flow Engine.

The report is displayed in vi.

15. To exit the viewer, type `:q!` and press Return.
16. Click OK in the Implementation Status dialog to return to the Xilinx Design Manager.

Note: To use another text editor, such as Emacs, as the report viewer, select **File** → **Preferences** from the Xilinx Design Manager.

Timing Simulation

For HDL designs, the Xilinx Design Manager produces a V (Verilog) file or a VHDL file expressed in SIMPRIMs and a corresponding SDF file that contains the timing data. The SDF file for VHD and V files are not interchangeable since the models they annotate follow different modeling standards.

Compiling the SIMPRIM Libraries

Before performing timing simulation on an HDL-based design, the VHDL or Verilog SIMPRIM libraries must be compiled with `qvhcom/qvlcom`. Your system administrator should perform this during installation. Perform these steps:

1. Verify that the libraries have been compiled.

The path to the VHDL libraries should be:

```
$XILINX/mentor/data/vhdl/simprim
```

The path to the Verilog libraries should be:

```
$XILINX/mentor/data/verilog/simprim
```

2. If these compiled SIMPRIM Libraries do not exist, contact your systems administrator. The Mentor Graphics Installation section of the *Alliance Installation Guide* describes how to compile the SIMPRIM libraries.

Passing Timing Generics to Special Cells—ROC, OSC, OSC4, and OSC5

If your designs do not have an external global set or reset port or a user defined internal net driving the global set/reset net, then a ROC (Reset on Configuration) cell is automatically added to your VHDL

netlist. This cell enables you to toggle the global set/reset net at the beginning of simulation by defining the pulse width of the signal pulse starting at time 0. By default, the pulsewidth is 0 which enables simulation to proceed but does not reset the circuit. To properly simulate the reset behavior of the chip, the pulse width generic should be set to a value within the range found in the Xilinx Databook for the particular device.

You can modify the following configuration for the technology's specific pulse width and user's testbench and compile it before you compile the testbench.

```
CONFIGURATION cfg_my_timing_testbench OF my_testbench IS
  FOR my_testbench_architecture
    FOR ALL:my_design USE ENTITY work.my_design(structure);
    FOR structure
      FOR ALL:roc USE ENTITY work.roc(roc)v)
        GENERIC MAP ( width => 100 ms);
      END FOR;
    END FOR;
  END FOR;
END FOR;
END cfg_my_timing_testbench;
```

Verilog designs do not require ports to drive the global/set reset net from a testbench. Therefore, Verilog designs do not contain the ROC cell. The same signal name found in the front end can be used to drive the signal in the back-annotated design. The signal must be driven, or all flip-flops will initialize as X.

VHDL designs that contain oscillator cells like OSC, OSC4, or OSC5, must have the clock period set with a configuration statement. By default, the period is 0, disabling the oscillator. You should carefully select the period from the range of viable periods found in the *Xilinx Databook* for the particular technology. A specific period is not guaranteed because the cell is subject to process variations. You should select the value that best meets your simulation requirements.

You can use the following configurations for either the OSC, OSC4, or OSC5 cells by just changing the name of the cell and modifying the pulse width to the correct value.

```
CONFIGURATION cfg_my_functional_testbench OF my_testbench IS
  FOR my_testbench_architecture
    FOR ALL:my_design USE ENTITY work.my_design(my_design_rtl);
    FOR my_design_rtl
```



```

FOR ALL:my_submodule USE ENTITY work.my_submodule(my_submodule_rtl);
FOR my_submodule_rtl
  FOR all: osc4 USE ENTITY work.osc4(structure)
    GENERIC MAP ( period_8m => 125 NS);
  END FOR;
END FOR;
END FOR;
END FOR;
END FOR;
END FOR;
END cfg_my_testbench_functional;

```

You can drive Verilog designs by the signal name used to drive the front-end simulation since the hierarchical name is preserved.

Compiling the Design

Before performing timing simulation on an HDL-based design, you must compile your VHDL or Verilog modules with `qvhcom`/`qvlcom`.

1. Create a map and working library with `qhlib` and `qhmap`.

```
qhlib work time_sim_lib
```

```
qhmap work
```

You should compile to a different work library than the one used for functional simulation to avoid data integrity mishaps.

2. Use `QHMAP` to add a `SIMPRIM` library listing in the QuickHDL `.ini` file:

```
qhmap simprim compiled_simprim_area
```

The locations of the compiled `simprim` libraries (`compiled_simprim_area`) are normally as follows:

```
$XILINX/mentor/data/vhdl/simprim
```

```
$XILINX/mentor/data/verilog/simprim
```

3. Type the following on the UNIX command line:

```
qvhcom [options] time_sim.vhd (VHDL)
```

```
qvlcom [options] tim_sim.v (Verilog)
```

Design_name is the name of the VHDL or Verilog file produced by the Xilinx Design Manager. See the Mentor Graphics documentation for information on the options available.

4. Compile any required configurations for special cells like ROC (reset on configuration) or OSC (see the “[Passing Timing Generics to Special Cells—ROC, OSC, OSC4, and OSC5](#)” section in this chapter).

```
qvhcom [options] configuration_file (VHDL)
```

```
qvlcom [options] configuration_file (Verilog)
```

5. Select the appropriate architecture configuration or module for your testbench and select QHDL in the pld_dmgr tools window.

See the Mentor documentation for QuickHDL instructions.

This procedure creates HDL database files that you can submit to QuickHDL.

Simulating the Design

Simulate with QuickHDL using qhsim. To include the timing information in the SDF file, invoke qhsim with the -sdftyp option. Refer to the Mentor documentation for information on available options. To simulate a Verilog based design, invoke qhsim with the -L simprim option to choose the Verilog simprim libraries models.

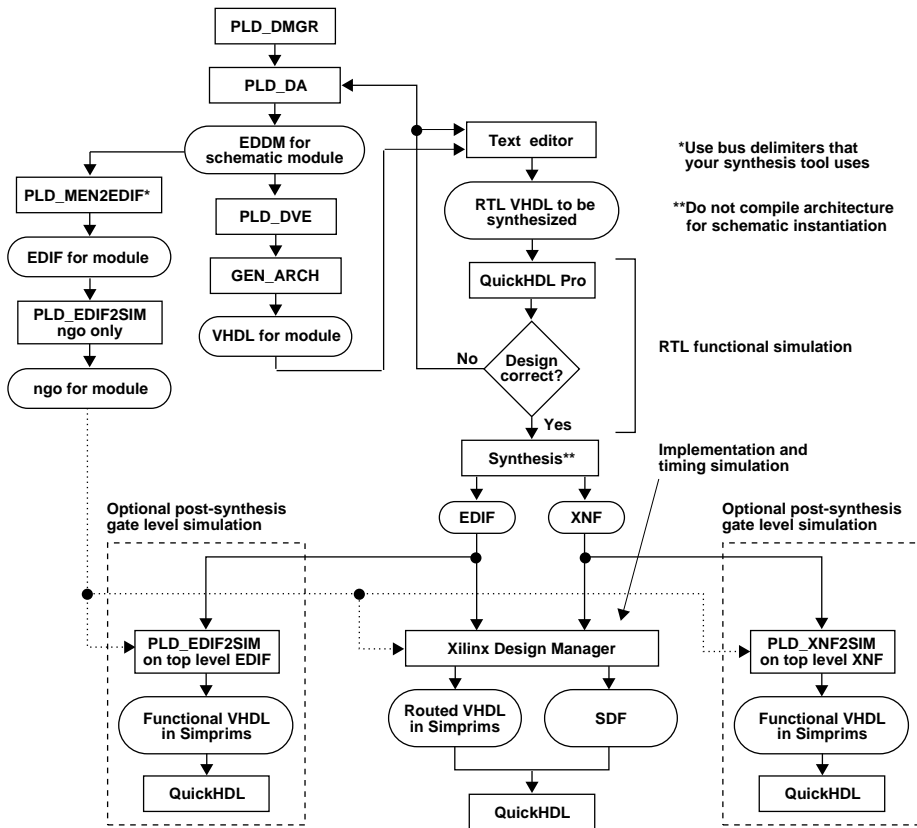
Mixed Designs with VHDL on Top

This chapter describes how to use the Mentor Graphics Interface to design with mixed schematic and VHDL designs with VHDL on Top. It contains the following sections:

- “The Design Flow” section
- “Design Entry” section
- “Functional Simulation” section
- “Design Implementation” section
- “Timing Simulation” section

The Design Flow

The design flow for a top-level VHDL design with a schematic sub-module embedded within is illustrated in the following figure.



X8027

Figure 5-1 Mixed Schematic and VHDL Design with VHDL on Top

Design Entry

Enter your pure VHDL design as described in the **“HDL Design Entry”** section of the **“HDL Designs”** chapter.

If you wish to insert a schematic module into your VHDL code, Mentor QuickHDL Pro allows you to co-simulate your VHDL portion in QuickHDL with your schematic portion in QuickSim II.

Your synthesizer requires you to treat the schematic module as a black box. You must use `pld_men2edif` and `pld_edif2sim` to create a NGO file for the schematic component so the Xilinx implementation tools can merge it in the module during implementation.

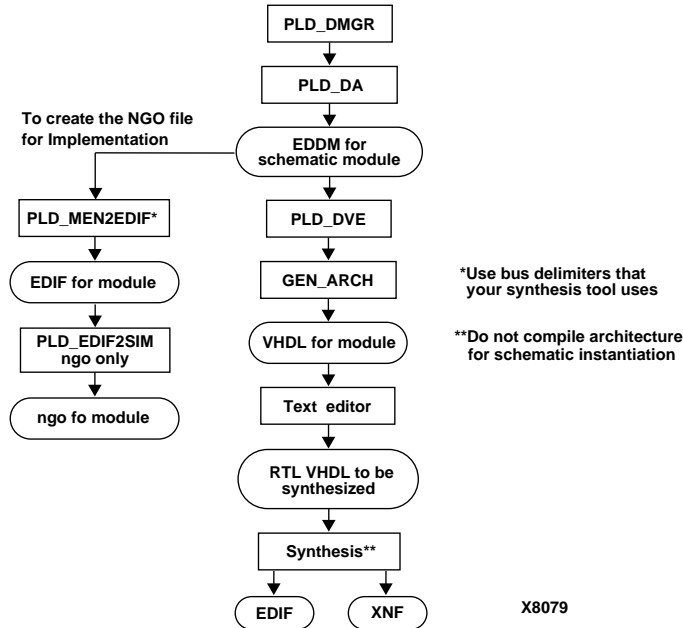


Figure 5-2 Design Entry for a Mixed Schematic and VHDL Design with VHDL on Top

To enter a mixed schematic and HDL design with VHDL on top, perform the following procedure. The “**Design Entry for a Mixed Schematic and VHDL Design with VHDL on Top**” figure shows the flow diagram for this procedure.

1. Open `pld_dmgr`.
2. Open `pld_da` and generate EDDM for the schematic module.
3. Create the NGO file for implementation. To accomplish this, you use `pld_men2edif` to convert the EDDM for schematic module to EDIF and then use `pld_edif2sim` to create the NGO file. The procedure for doing this is as follows:

- a) Open pld_men2edif.

A dialog box opens as shown in the following figure.

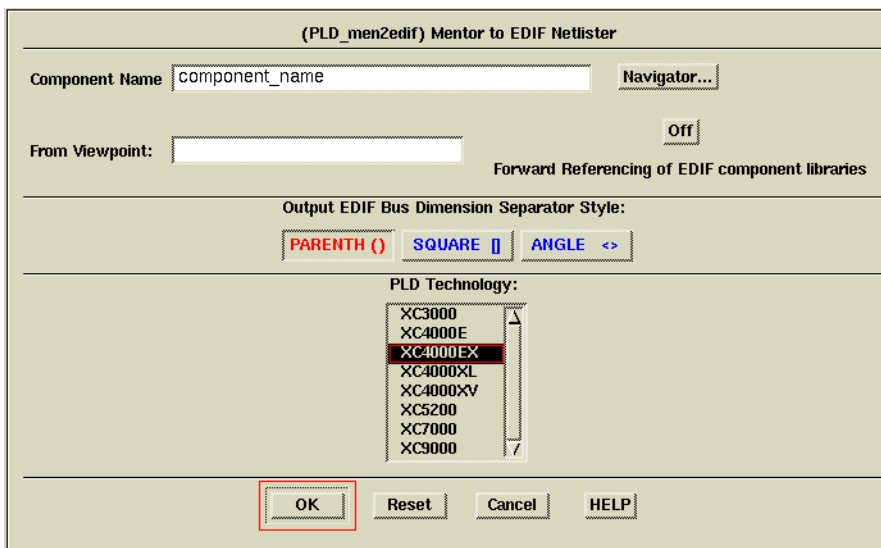


Figure 5-3 Mentor to EDIF Netlister Dialog Box

- b) Fill in the component name of the existing schematic based module. The module must have a symbol for its top-level netlist. There can be no chip-level I/Os.
- c) Select a viewpoint that properly sets the schematic parameters such that the EDIF is properly generated.
- d) Select the Bus Dimension Separator Style that matches your synthesizer. This is important; if your synthesizer uses one bus style and the EDIF/NGO from your schematic uses another style, the implementation tool does not merge the schematic module with the rest of the design, thus leaving it unexpanded.
- e) Choose the technology.
- f) Click OK.
- g) Create the NGO from EDIF2SIM and XNF2SIM for later use in the implementation tool. EDIF2SIM and XNF2SIM NGO files must be placed in your top level directory or you must

modify the macro search path in the Xilinx Design Manager to include the location of the NGO files. EDIF2SIM or XNF2SIM do not have the macro search path functionality. You must have the EDIF2SIM and XNF2SIM NGO files in the same directory as your top-level EDIF or XNF.

h) Open `pld_edif2sim`.

The dialog box opens as shown in the following figure.

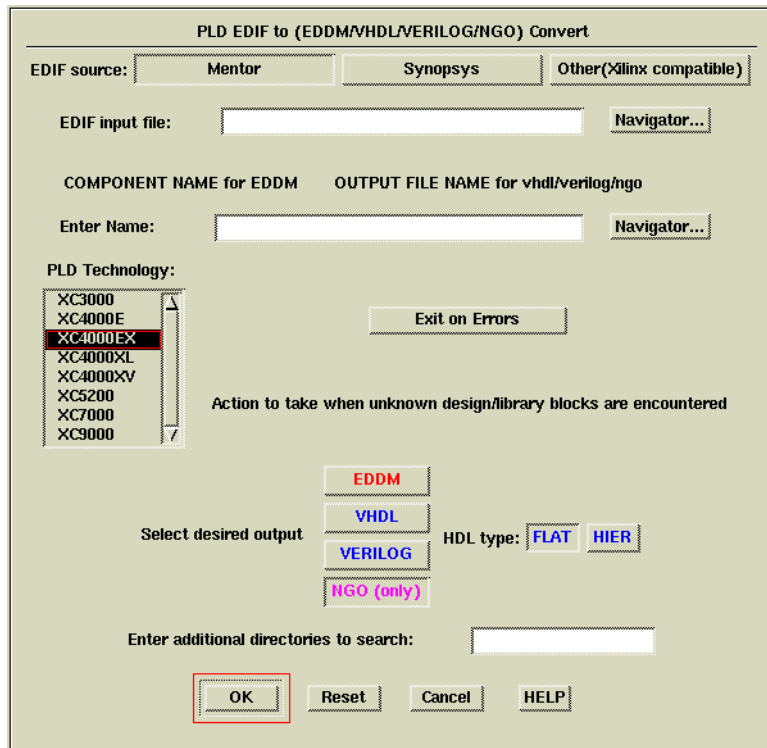


Figure 5-4 Pld_edif2sim Dialog Box

- i) Specify the source of the EDIF file as either a Mentor, Synopsys, or Xilinx compatible EDIF. This step selects the appropriate implementation libraries.
- j) Enter the name for the EDIF file created above in step f that will be used for the NGO file.

- k) Enter the name of the NGO file based on the component name used in the VHDL instantiation.
 - l) Select a Xilinx technology.
 - m) Select the NGO (only) output.
 - n) In the “Enter additional directories to search” field, enter all the directory pathnames that the program should search to find supporting EDIF, XNF, and NGO files.
 - o) Click OK to produce the NGO macro file of the schematic component.
4. Use `pld_dve` to set the simulation viewpoint.
 5. Open `GEN_ARCH` to generate the VHDL for module.
- The dialog box opens as shown in the following figure.

Create a VHDL Architecture from an Eddm component

Eddm component Navigator...

Source directory Navigator...

VHDLWrite Options file Navigator...

QuickHDL initialization file Navigator...

DVE Dofile Navigator...

Library name Navigator...

Model label

Get VHDL Leaves

Nocompile:

Only generate architecture:

Replace entity:

Verbose output:

OK Reset Cancel

Figure 5-5 Create a VHDL Architecture from an EDDM Component Dialog Box

6. Enter the EDDM component name for the schematic.
7. Indicate the directory where the VHDL source files from GEN_ARCH are to be placed.
8. Specify the appropriate QuickHDL initialization file. See the Mentor Graphics Documentation for details.
9. Enter the library name in which the compiled code will be placed. You can place it in the work library.
10. Leave the other boxes blank and click OK to produce the required output.

11. Use a Text Editor to create RTL VHDL to be synthesized for the rest of the design. Include the component declaration and instantiation for the schematic module.
12. Perform synthesis to generate EDIF or XNF for the whole design with a black box for the schematic module.

Functional Simulation

VHDL-on-top designs consist of a VHDL based design referencing EDDM components.

Compiling the Design

Before functionally simulating the mixed VHDL-based design, perform the following steps:

1. Create a working library with `qhlib`.

```
qhlib work
```

2. Map the library with `qhmap`.

```
qhmap work work
```

3. If using LogiBLOX modules, use `qhmap` to map to the compiled LogiBLOX modules location.

```
qhmap logiblox $XILINX/mentor/data/vhdl/logiblox
```

4. Compile the VHDL source files with `qvhcom`.

```
qvhcom [options] design_name
```

See the Mentor documentation for a description of the available options.

Simulating the Design

To simulate VHDL-at-top designs, invoke QuickHDL Pro, which in turn invokes QuickSim to simulate the Unified Libraries elements and QuickHDL to simulate the VHDL-based blocks as needed.

1. Double-click the left mouse button on the QuickHDL Pro icon in the Design Manager Tools window.

Alternatively, you can select the top-level component in the Navigator window and click the right mouse button to invoke QuickHDL Pro.

The QHDL Pro dialog box appears, as shown in the “QHDL Pro Dialog Box” figure.

2. In the **Invoke On** field, click on **Configuration**.
3. In the **Name** field, type the path name of the configuration from Gen_Arch.
4. Click on **Qhpro**.
5. Click on **OK** to proceed with simulation.

For details on using QHDL Pro, refer to the Mentor Graphics Documentation.

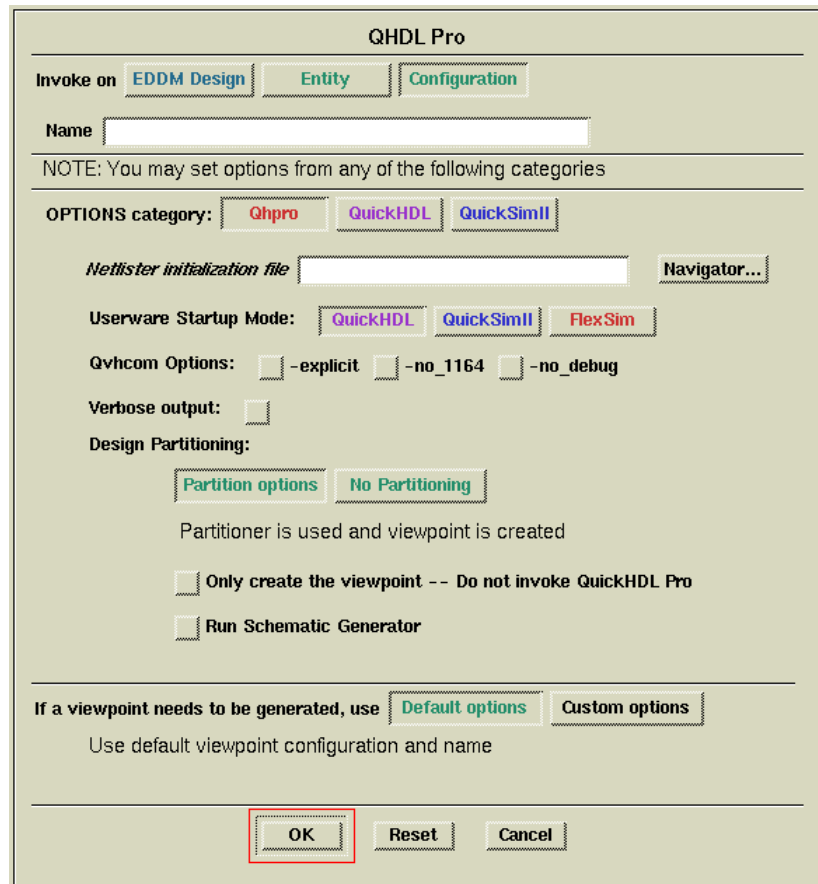


Figure 5-6 QHDL Pro Dialog Box

Optional Post-Synthesis Functional Simulation

You can optionally re-simulate the design after synthesis to an EDIF or XNF file to ensure that the design's functionality remains optimal. To do so, follow these steps:

1. Create the NGO from EDIF2SIM and XNF2SIM for later use in the implementation tool. EDIF2SIM and XNF2SIM NGO files must be placed in your top level directory or you must modify the macro search path in the Xilinx Design Manager to include the location of NGO files. EDIF2SIM or XNF2SIM do not have the

macro search path functionality. You must have the EDIF2SIM and XNF2SIM NGO files in the same directory as your top-level EDIF or XNF.

2. If the synthesis tool created an EDIF file, submit the file to `pld_edif2sim`, then submit it to QuickHDL.
3. If the synthesis tool created an XNF file, submit the file to `pld_xnf2sim`, then submit it to QuickHDL.

Design Implementation

Once you complete the functional simulation and synthesis steps for a VHDL-on-top design, you are ready to implement your design in an FPGA or CPLD. You perform implementation with the Xilinx Design Manager, a graphical design flow and project manager. In the Mentor interface, the Xilinx Design Manager is called `pld_dsgnmgr`. You invoke `pld_dsgnmgr` from the Mentor Design Manager or from a UNIX shell.

Design entry of VHDL-on-top designs produces NGO files for schematic modules and XNF or EDIF files for the synthesized portion of the design. The following figure shows the design flow for implementing such a mixed design.

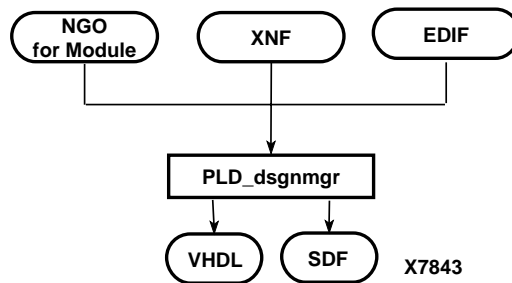


Figure 5-7 Design Implementation

The Xilinx Design Manager takes in your design, represented by the EDIF or XNF file from synthesis and the NGO file for the schematic module from `pld_edif2sim`. It first translates the design into a flattened or hierarchical netlist, then optimizes, places, and routes the

design. You can also use the Xilinx Design Manager to generate SDF timing information that you can import into QuickHDL. For a more in-depth discussion of the flow, including advanced implementation options, see the *Development System User Guide*.

By default, the Xilinx Design Manager looks for the NGO files for the schematic modules in the directory where it was invoked. You have the option of putting all of the NGO files in another directory. To direct the Xilinx Design Manager to look for the NGO files in another directory, follow these steps:

1. In the Xilinx Design Manager window, select **Utilities** → **Template Manager**.
2. Select the Family for implementation.
3. Select Implementation Templates.
4. Select the Template you wish to modify.

If you have not created your own template, you may modify the default one.

5. Select Edit.
6. Select Interface.
7. Fill in the Macro Search Path box with the path to the NGO files.
8. Under simulation Data Options, select the VHDL Format as shown in the following figure.

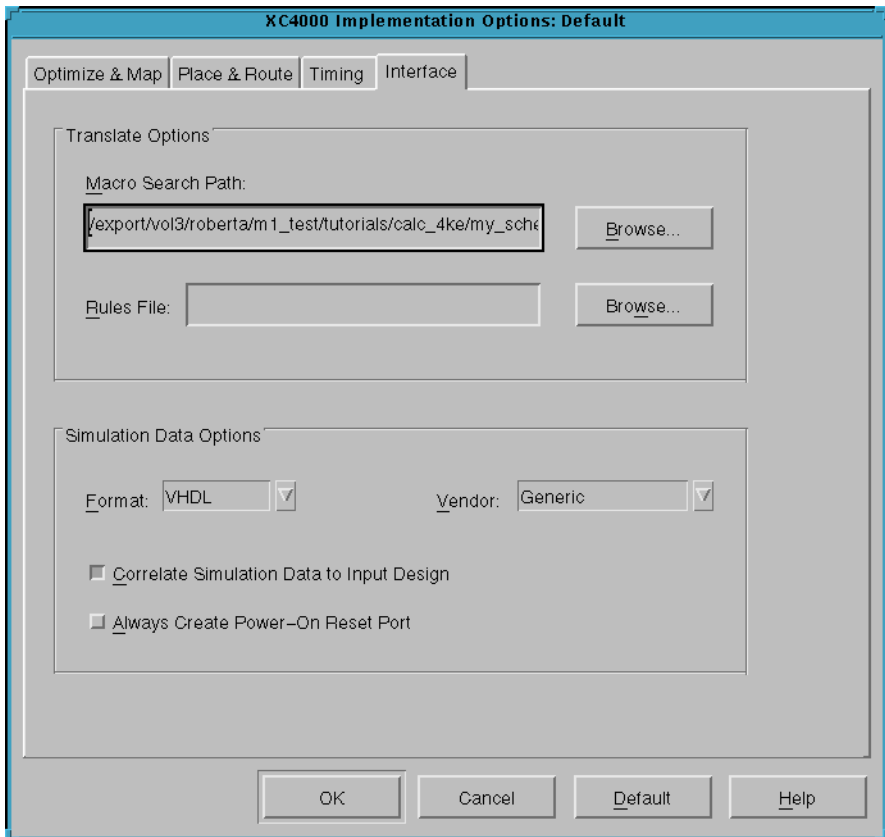


Figure 5-8 Implementation Options Dialog Box

To implement your design follow these steps:

1. Within the Mentor Design Manager, select the EDIF icon for your design in the Navigator, then select **Right Mouse Button** → **Open** → **p1d_dsgnmgr**. The Xilinx Design Manager appears as shown in the “Xilinx Design Manager” figure. The tool automatically creates a Xilinx project called *your_design_name*. Xilinx project information is kept in a file called `xproj/your_design_name.prj` by default.

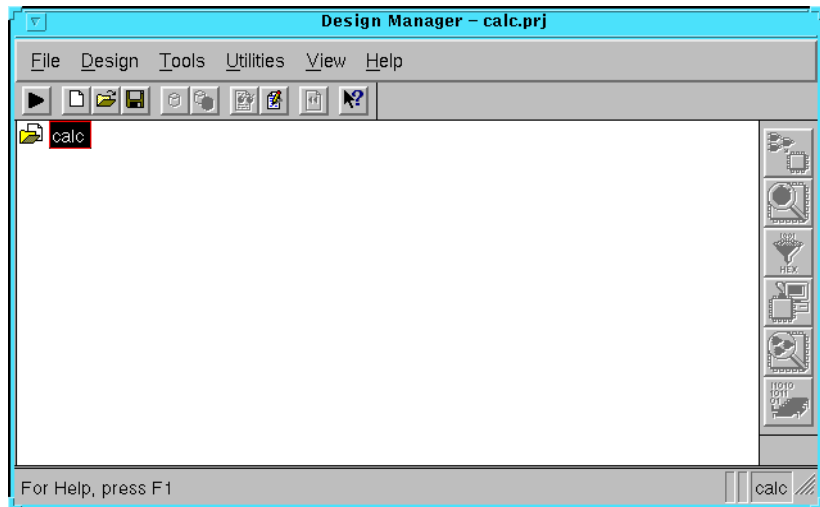


Figure 5-9 Xilinx Design Manager

Each project is associated with objects known as *versions* and *revisions*. Versions represent logic changes in a design (for example, adding a new block of logic, replacing an AND gate with an OR gate, or adding a flip-flop); revisions represent different executions of the design flow on a single design version, usually with new implementation options (for example, higher place and route effort, a change in part type, or experimentation with new bitstream options).

2. Within the Xilinx Design Manager, select **Design** → **Implement**.

The Implement dialog box opens as shown in the following figure and displays fields for part type, design version, and revision.

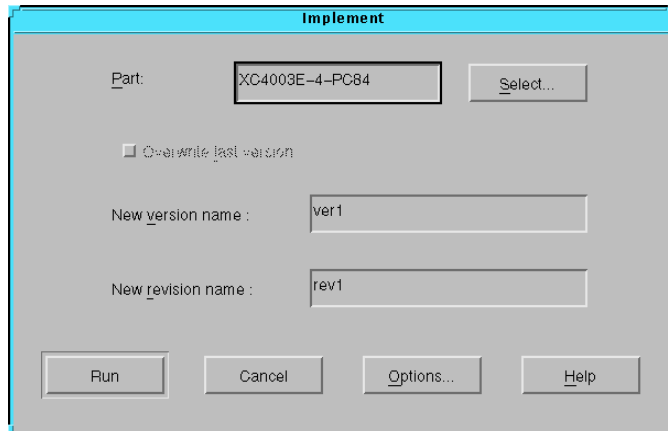


Figure 5-10 Implementation Dialog Box

3. The Xilinx Design Manager reads the part type from the design.

If you wish to specify the part type manually, click the Select button to display a pull-down listing of available devices. Choose a family, a device, a package, and a speed grade. Click OK. The part number is inserted into the Part field in the Implement dialog box.

4. Click on Options. The Options dialog box appears as shown in the **“Options Dialog Box”** figure.

Note: The CPLD Options dialog box does not have a Configuration Template section, nor does it have a Produce Logic Level Timing Report checkbox.

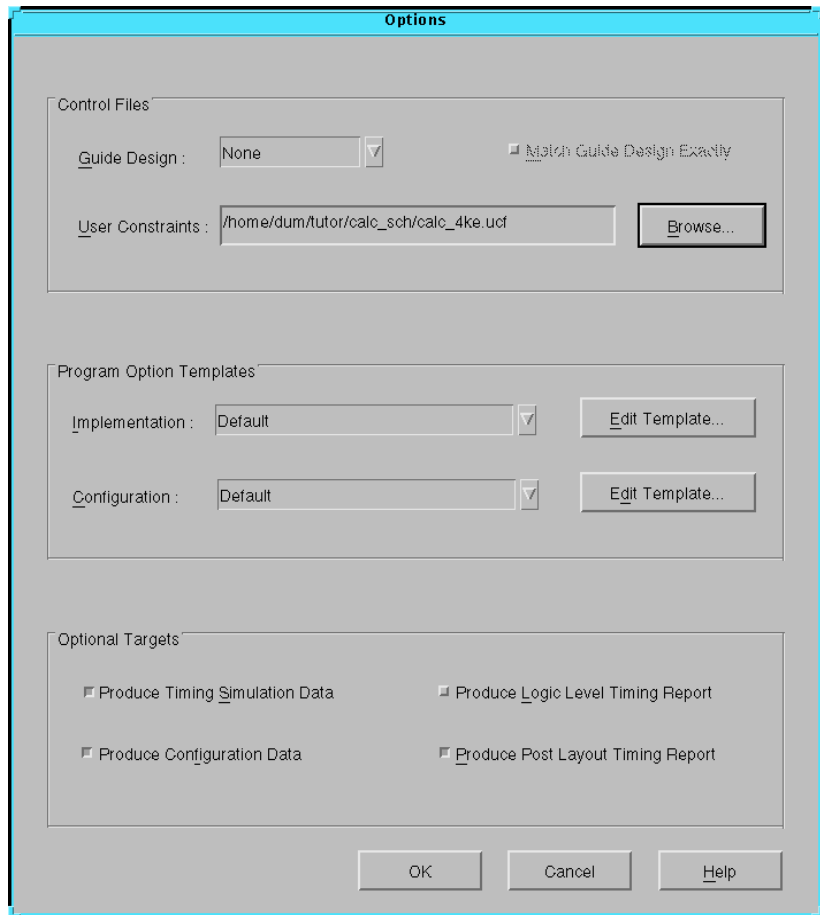


Figure 5-11 Options Dialog Box

5. Click the Browse button next to the User Constraints field. Select the appropriate .ucf file from the design directory, then Click OK.
6. You may use your own template for implementation or configuration. For instance, you might have an implementation where the macro search path template has been set. If so, select the proper template under Program Option Templates.
7. Under Optional Targets, make sure the following are selected:

- **Produce Timing Simulation Data:** This generates a back-annotated VHDL file that you can import into the Mentor Graphics tools.
- **Produce Configuration Data:** This generates a programming bitstream suitable for downloading into the Xilinx device.
- **Produce Post Layout Timing Report:** This generates a timing report file based on how the design is actually routed.

You can also select the following option (FPGAs only):

- **Produce Logic Level Timing Report:** This generates a preliminary (pre-place and route) timing report based on the number of logic levels in each signal path. Since it is generated before the place-and-route layout step, it does not contain information on device routing. Looking at this report before place and route can be useful for seeing how much routing slack you have in a design.
8. Select the **Edit Template** button on the right hand side of the Implementation field. The Implementation Options dialog box appears as shown in the following figure.

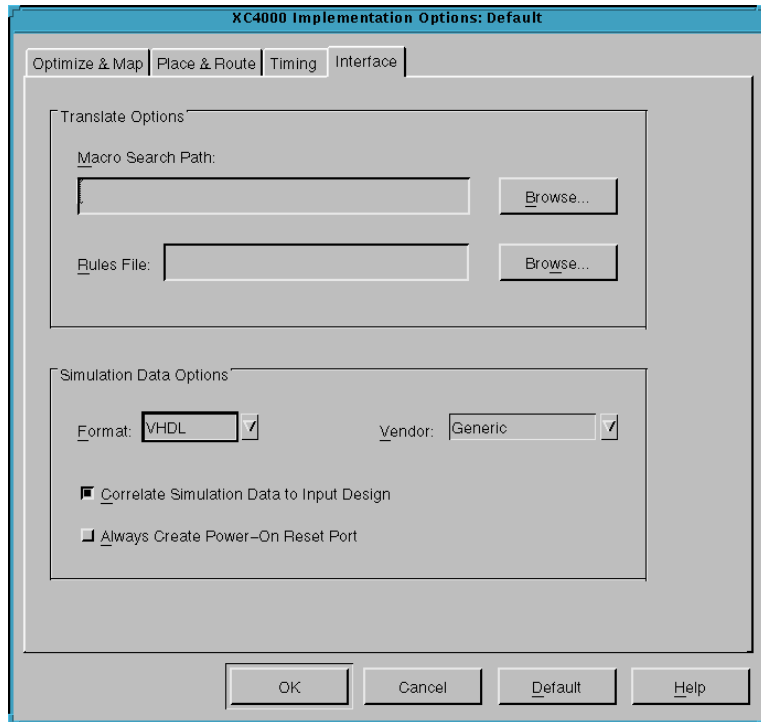


Figure 5-12 Changing Simulation Data Option

9. Select the Interface tab. In the Interface pane, look under Simulation Data Options and verify that Format is set to VHDL and that Correlate Simulation Data to Input Design is selected. In the Vendor field, select generic.
10. Click OK to return to the Options window.
11. Click OK to return to the Implementation dialog box.
12. In the Xilinx Design Manager window, verify that you have selected the current version and revision you wish to work on, then click Run. The Flow Engine comes up as shown in the following figure.

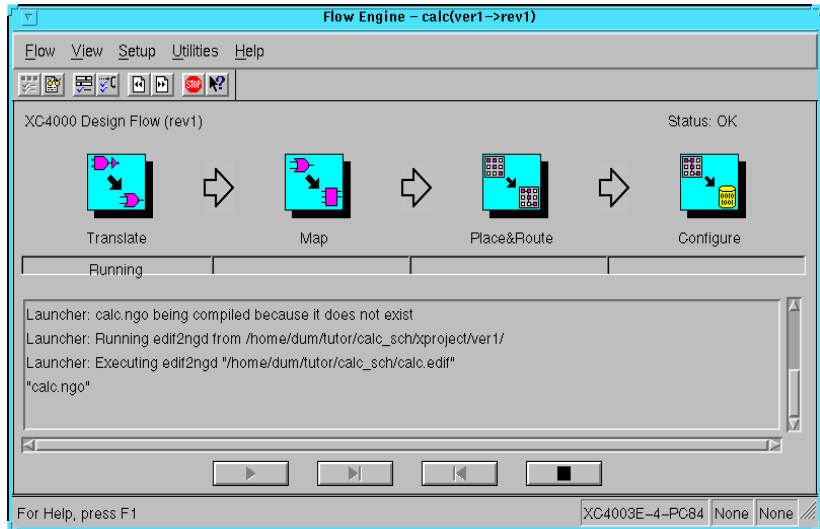


Figure 5-13 The Xilinx Flow Engine

The status bar shows the progress of the implementation flow with the following stages:

- **Translate:** Converts the design EDIF or XNF file into an NGD (Native Generic Design) file.
- **Map:** Groups basic elements (bels) such as flip-flops and gates into logic blocks (comps). Also generates a logic-level timing report if desired.
- **Place&Route:** Places comps into the device, and routes signals between them.
- **Timing:** Generates timing simulation data and an optional post-layout timing report.
- **Configure:** generates a bitstream suitable for downloading into and configuring a device

When the implementation completes, an Implementation Status box appears with:

```

Implementing revision ver1->rev1 completed
successfully.

```

13. Click on **View Logfile** to display the logfile from Flow Engine in the vi text editor.
14. To exit the viewer, type **:q!** and press Return.
15. Click **OK** in the Implementation Status dialog to return to the Xilinx Design Manager.

Note: To use another text editor, such as Emacs, as the report viewer, select **File** → **Preferences** from the Xilinx Design Manager.

For VHDL-based designs, the Xilinx Design Manager produces a VHDL file and a SDF file that expresses timing and simulation in SIMPRIM library elements instead of Unified Libraries elements.

Timing Simulation

You can now submit the VHDL and SDF files to QuickHDL for timing simulation. There is no longer a need to use QuickHDL Pro.

Compiling the SIMPRIM Libraries

Before performing timing simulation on an HDL-based design, the VHDL SIMPRIM libraries must be compiled with qvhcom. Your system administrator should perform this during installation. Perform these steps:

1. Verify that the libraries have been compiled.

The path to the VHDL libraries should be:

```
$XILINX/mentor/data/vhdl/simprim
```

2. If these compiled SIMPRIM Libraries do not exist, contact your systems administrator. The Mentor Graphics Installation section of the *Alliance Installation Guide* describes how to compile the SIMPRIM libraries.

Passing Timing Generics to Special Cells—ROC, OSC, OSC4, and OSC5

If your designs do not have an external global set or reset port or a user defined internal net driving the global set/reset net, then a ROC (Reset on Configuration) cell is automatically added to your VHDL netlist. This cell enables you to toggle the global set/reset net at the beginning of simulation by defining the pulse width of the signal

pulse starting at time 0. By default, the pulsewidth is 0 which enables simulation to proceed but does not reset the circuit. To properly simulate the reset behavior of the chip, the pulse width generic should be set to a value within the range found in the Xilinx Databook for the particular device.

You can modify the following configuration for the technology's specific pulse width and user's testbench and compile it before you compile the testbench.

```
CONFIGURATION cfg_my_timing_testbench OF my_testbench IS
  FOR my_testbench_architecture
    FOR ALL:my_design USE ENTITY work.my_design(structure);
    FOR structure
      FOR ALL:roc USE ENTITY work.roc(roc)v)
        GENERIC MAP ( width => 100 ms);
      END FOR;
    END FOR;
  END FOR;
END FOR;
END cfg_my_timing_testbench;
```

Verilog designs do not require ports to drive the global/set reset net from a testbench. Therefore Verilog designs do not contain the ROC cell. The same signal name found in the front end can be used to drive the signal in the back-annotated design. The signal must be driven, or all flip-flops will initialize as X.

VHDL designs that contain oscillator cells like OSC, OSC4, or OSC5, must have the clock period set with a configuration statement. By default, the period is 0, disabling the oscillator. You should carefully select the period from the range of viable periods found in the Xilinx Databook for the particular technology. A specific period is not guaranteed because the cell is subject to process variations. You should select the value that best meets your simulation requirements.

You can use the following configurations for either the OSC, OSC4, or OSC5 cells by just changing the name of the cell and modifying the pulse width to the correct value.

```
CONFIGURATION cfg_my_functional_testbench OF my_testbench IS
  FOR my_testbench_architecture
    FOR ALL:my_design USE ENTITY work.my_design(my_design_rtl);
    FOR my_design_rtl
      FOR ALL:my_submodule USE ENTITY work.my_submodule(my_submodule_rtl);
      FOR my_submodule_rtl
```

```
FOR all: osc4 USE ENTITY work.osc4(structure)
  GENERIC MAP ( period_8m => 125 NS);
END FOR;
END FOR;
END FOR;
END FOR;
END FOR;
END FOR;
END cfg_my_testbench_functional;
```

You can drive Verilog designs by the signal name used to drive the front-end simulation since the hierarchical name is preserved.

Compiling the Design

Before performing timing simulation on an HDL-based design, you must compile your VHDL modules with qvhcom.

1. Create a map and working library with qhlib and qhmap.

```
qhlib work time_sim_lib
```

```
qhmap work
```

You should compile to a different work library than the one used for functional simulation to avoid data integrity mishaps.

2. Use QHMAP to add a SIMPRIM library listing in the QuickHDL .ini file:

```
qhmap simprim compiled_simprim_area
```

The locations of the compiled simprim libraries (*compiled_simprim_area*) are normally as follows:

```
$XILINX/mentor/data/vhdl/simprim
```

3. Type the following on the UNIX command line:

```
qvhcom [options] time_sim.vhd
```

Design_name is the name of the VHDL file produced by the Xilinx Design Manager. See the Mentor Graphics documentation for information on the options available.

4. Compile any required configurations for special cells like ROC (reset on configuration) or OSC (see the [“Passing Timing Generics to Special Cells—ROC, OSC, OSC4, and OSC5”](#) section in this chapter).

`qvhcom [options] configuration_file`

5. Select the appropriate architecture configuration or module for your testbench and select QHDL in the `pld_dmgr` tools window.

See the Mentor documentation for QuickHDL instructions.

This procedure creates HDL database files that you can submit to QuickHDL.

Simulating the Design

Simulate with QuickHDL using `qhsim`. To include the timing information in the SDF file, invoke `qhsim` with the `-sdftyp` option. Refer to the Mentor documentation for information on available options.

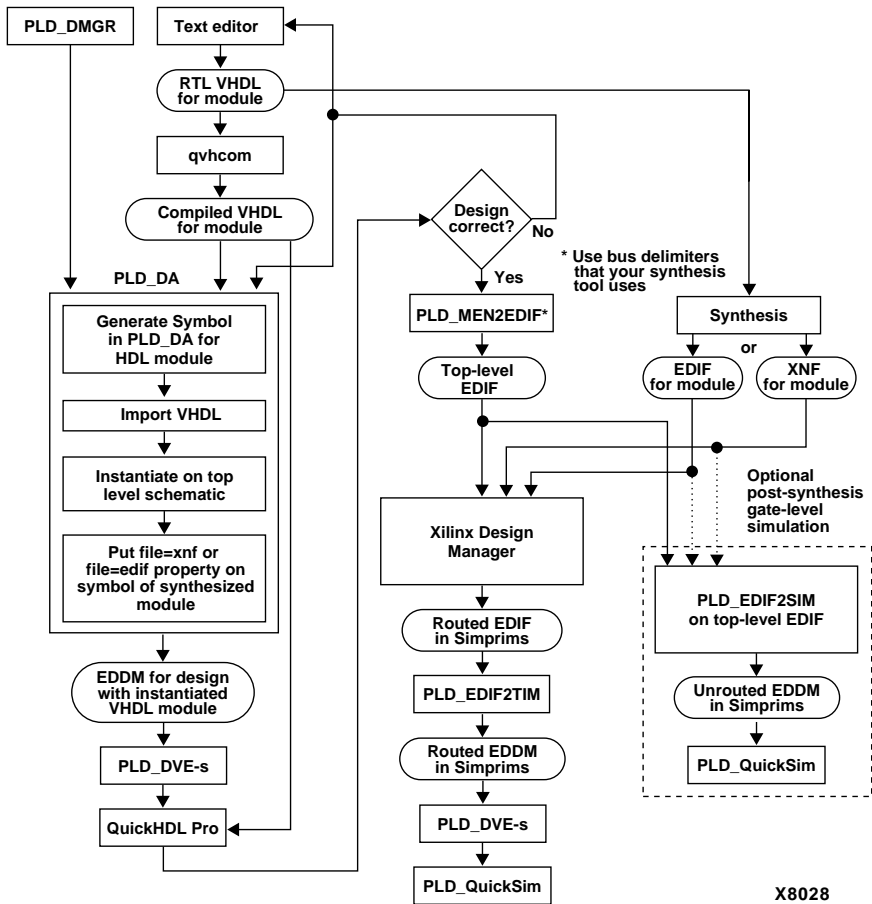
Mixed Designs with Schematic on Top

This chapter describes how to use the Mentor Graphics Interface to design with mixed schematic and VHDL designs with schematic on top. It contains the following sections:

- “The Flow” section
- “Design Entry” section
- “Functional Simulation” section
- “Design Implementation” section
- “Timing Simulation” section

The Flow

The design flow for designs containing a mixture of schematics and VHDL is illustrated in the following figure.



X8028

Figure 6-1 Mixed Schematic and VHDL Design with Schematic on Top

Design Entry

Design entry consists of two parts, VHDL module design entry and schematic entry.

VHDL Module Design Entry

To enter the VHDL module of your design and to get it ready for functional simulation and implementation, perform the following steps:

1. Enter the VHDL portion of your design as described in the “HDL Design Entry” section of the “HDL Designs” chapter.
2. When you have the RTL description for the module(s), create a working directory for the VHDL description.

```
qhlib work
```

Note: If you map to a work library other than the default work library, map the library with qhmap as follows:

```
qhmap work mywork
```

3. Compile the VHDL source files with qvhcom.

```
qvhcom [options] design_file(s) -qhpro_syminfo
```

4. In pld_da, use **File** → **Generate** → **Symbol** to import VHDL and create a symbol for the VHDL module as shown in the following figure.

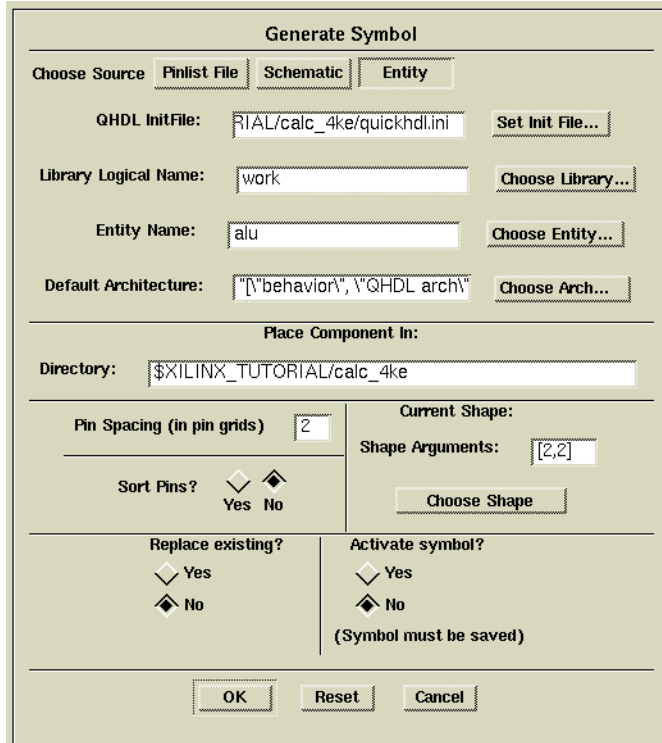


Figure 6-2 Generate Symbol Dialog Box

5. On the symbol, add the `file=xnf_file_pathname` or `file=edif_file_pathname` property with a value that specifies the path to the XNF or EDIF file that will be synthesized from the RTL description you created above.
6. Check and save the new symbol.

Refer to the Mentor documentation for details on using Generate Symbol.

Schematic Entry

1. Enter the top-level and lower-level schematic portions as described in the **“Design Entry”** section of the **“Schematic Designs”** chapter.

2. Instantiate the symbol created for the VHDL module on the top-level design.

Functional Simulation

Mixed-model schematic-based designs can be composed of schematic elements from the Unified Libraries, VHDL, XNF-based components, or EDIF-based components. The VHDL-based components will later have `FILE=edif_path` properties for implementation.

You can simulate the design either before or after you synthesize the HDL module.

Functional Simulation Before Synthesis

The flow diagram for this procedure is shown in the “**Performing Functional Simulation Before Synthesis on Mixed-Model Schematic-on-Top Designs**” figure. Follow these steps to simulate your design before you synthesize it:

1. Generate a symbol for the HDL module with `pld_da`.
2. Instantiate the symbol on the schematic.
3. Put `FILE=xnf_file_pathname` or `FILE=edif_file_pathname` property on the symbol of the synthesized module.
4. Create a viewpoint for the top-level design using `pld_dve`:

```
pld_dve -s design_name technology [viewpoint_name]
```

5. Run QuickHDL Pro to simulate the design by typing the following syntax:

```
qhpro [options] design_name
```

Alternate ways to invoke QuickHDL Pro are to double-click the left mouse button on the QuickHDL Pro icon in the Design Manager Tools window or to select the top-level component in the Navigator window and click the right mouse button.

The QHDL Pro dialog box appears, as shown in the following figure.

QHDL Pro

Invoke on **EDDM Design** **Entity** **Configuration**

Pathname **Navigator...**

Symbol Interface

NOTE: You may set options from any of the following categories

OPTIONS category: **Qhpro** **QuickHDL** **QuickSimII**

Netlist initialization file **Navigator...**

Userware Startup Mode: **QuickHDL** **QuickSimII** **FlexSim**

Qvhcom Options: -explicit -no_1164 -no_debug

Verbose output:

Design Partitioning:

Partition **No Partitioning**

Run the partitioner and then invoke QuickHDL Pro

If a viewpoint needs to be generated, use **Default options** **Custom options**

Use default viewpoint configuration and name

OK **Reset** **Cancel**

Figure 6-3 QHDL Pro Dialog Box

6. In this dialog box, click on **EDDM Design** in the Invoke On field.
7. In the Pathname field, type in the path name of the component.
8. Type the symbol name in the Symbol field only. This step is optional.
9. Type the interface name in the Interface field only. This step is optional.
10. Click **OK** to invoke the QuickHDL simulator and perform simulation.

11. After simulation, synthesize the HDL module with Synopsys' Design Compiler, Synopsys' FPGA Compiler, Exemplar Logic's Galileo, or another synthesizer that creates an EDIF or XNF file for Xilinx.

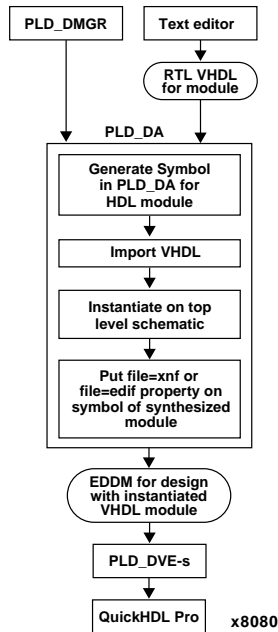


Figure 6-4 Performing Functional Simulation Before Synthesis on Mixed-Model Schematic-on -Top Designs

Functional Simulation After Synthesis

You can optionally re-simulate the design at this point to ensure that the design's functionality remains optimal. This method for simulating your design does not require the use of QuickHDL pro. The flow diagram for this procedure is shown in the **“Performing Functional Simulation After Synthesis on Mixed-Model Schematic-on -Top Designs”** figure.

If the synthesis tool created an EDIF file, you can include a symbol for the module within the top level design with `file=edif_file_name`. Then

submit the whole design to `p1d_edif2sim`, and then submit it to `p1d_quicksim`.

If the synthesis tool created an XNF file, you can include a symbol for the module within the top level design with `file=xnf_file_name`. Then submit the whole design to `p1d_edif2sim`, and then submit it to `p1d_quicksim`.

Follow these steps to simulate by this method:

1. Synthesize the HDL module that is being included on the schematic, and create an EDIF or XNF file from that synthesis.
2. Create a symbol for the HDL module with `p1d_da` and add the `file=edif_file_name` or `file=xnf_file_name` property to the symbol. Instantiate the symbol on the top level design.
3. Run `p1d_men2edif` on the top level design to create an EDIF for the whole design. Make sure to specify the appropriate bus delimiter to match the synthesized module.
4. Run `p1d_edif2sim` to convert it to a Mentor EDDM single object:

```
p1d_edif2sim edif_file symbol_component_name technology  
{-m|-s} -eddm
```

Use `-m` if the synthesis was performed with a Mentor tool; use `-s` if the synthesis was performed with a Synopsys tool.

5. Perform functional simulation with `p1d_quicksim`:

```
p1d_quicksim design_name[ /viewpoint_name]
```

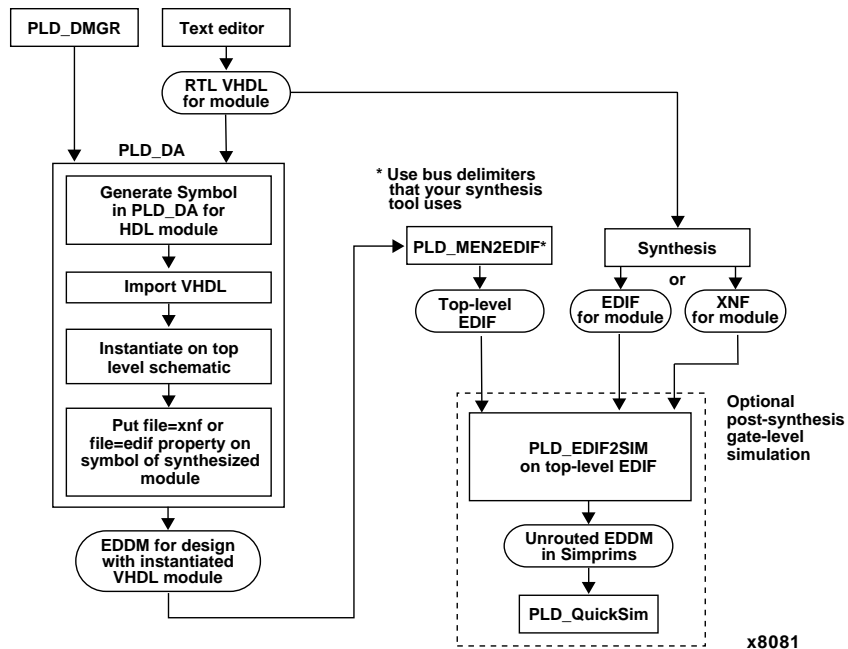


Figure 6-5 Performing Functional Simulation After Synthesis on Mixed-Model Schematic-on -Top Designs

Design Implementation

After functional simulation, use a synthesis tool that creates a Xilinx compatible EDIF or XNF file to synthesize certain blocks of the design described in VHDL.

After synthesis, you must attach a `FILE=design.edif` or `FILE=design.tnf` property to the VHDL-based block symbol in the schematic before you submit the top-level EDDM design to `pld_men2edif`.

Converting the EDDM Design

You convert the top-level EDDM design to EDIF with the `pld_men2edif` utility. To convert your design to EDIF, follow these steps.

1. In the Mentor Design Manager, double-click the left mouse button on the `p1d_men2edif` icon.

The dialog box shown in the following figure appears.

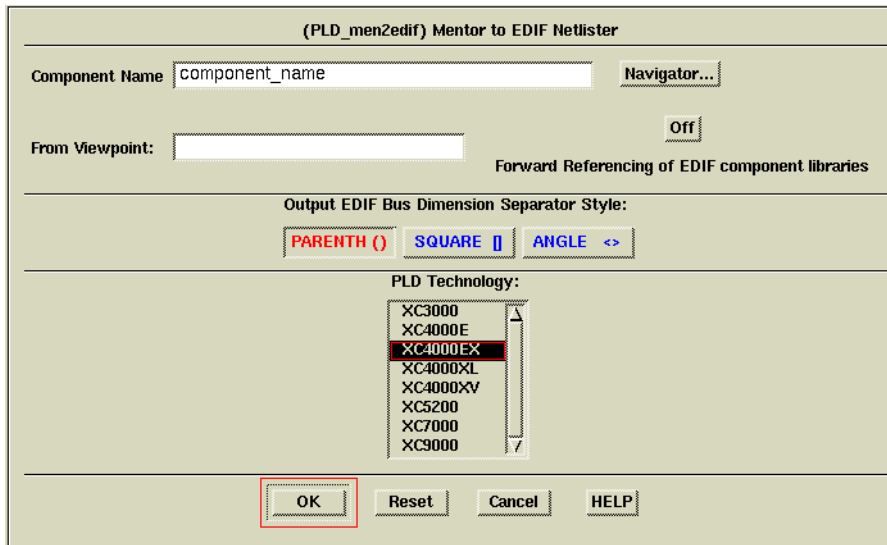


Figure 6-6 Mentor to EDIF Netlist Dialog Box

2. In the Component Name field, enter the component name, or click on **Navigator** to browse a list of design names.
3. In the From Viewpoint field, you can enter the viewpoint name if you do not want to use the default viewpoint. Alternatively, in step 2 you can select a viewpoint under the component.
4. Select the appropriate architecture for your design in the PLD Technology field.
5. Select the desired bus notation style.
Be careful to select the Bus Dimension Separator Style that matches your synthesizer's style. Otherwise busses between the schematic portion and the HDL portion will not match up in the implemented design.
6. Click on **OK**.

pld_men2edif now produces an EDIF file that you can submit to the Xilinx Design Manager, pld_dsgnmgr. The output name is *component_name.edif*.

Implementing the Design

The Xilinx Design Manager, pld_dsgnmgr, can accept an EDIF file, or if your design is a pure XNF design, it can accept an XNF file.

In the Mentor Design Manager, double-click the left mouse button on the pld_dsgnmgr icon.

Since the implementation is essentially the same as for a pure schematic design, follow the directions in the **“Implementing Schematic Designs”** section of the **“Schematic Designs”** chapter.

Normally you need an EDIF file to bring back into the EDDM environment. But you have the option of creating a VHDL or Verilog and an SDF file instead of an EDIF file, which you can submit to QuickHDL for timing simulation.

Timing Simulation

This is the same as the **“Timing Simulation for Schematic Designs”** section of the **“Schematic Designs”** chapter. When reading this section, be aware that cross-probing does not apply to the VHDL component.

Advanced Techniques

This chapter discusses aspects of schematic entry and simulation that you should be familiar with to use Design Architect and `pld_quicksim` effectively.

This chapter contains the following sections:

- “Retargeting the Design to a Different Family” section
- “Merging Design Files from Other Sources” section
- “Simulation Models” section
- “Analyzing Nets from the Schematic” section
- “Setting Global Reset and 3-State Signals” section

Retargeting the Design to a Different Family

The Unified Libraries allow you to retarget your designs from one device family to another if 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 across Xilinx device families.

The procedure described in the following section uses Xilinx’s Convert Design utility in Design Architect to retarget your schematic. It allows 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 XC3000A design that is retargeted for use in an XC4000E device, you must manually replace the GCLK symbol with a BUFGP, BUFG, or BUFGS, which is the XC4000E equivalent of a GCLK.

Note: In the following procedures, XC4000 is used as the source design device family, and XC5200 is used as the target design device family. You can also retarget other device families.

To retarget a design to a different family, perform these steps:

1. Activate Design Architect by using either of the methods described in the **“Invoking Design Architect” section of the “Schematic Designs” chapter**. You do not have to open the schematic.
2. On Design Architect’s desktop background (the area outside any schematic or symbol windows) press the right mouse button and select **Convert Design**.

The dialog box shown in the following figure appears.

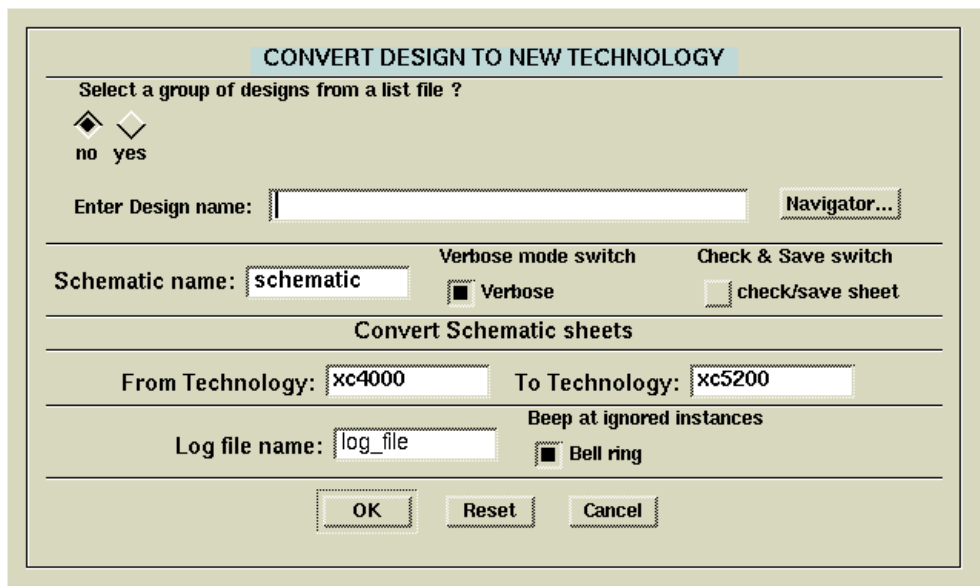


Figure 7-1 Convert Design To New Technology Dialog Box

3. In the field asking “Select a group of designs from a list file?,” click on **yes** or **no**.
 - Click **no** if you want to retarget a single design. Convert Design utility traverses the hierarchy of a given schematic and converts the schematics of any hierarchical blocks found on the top-level schematic.
 - Click **yes** if you have a number of designs to retarget, and their names are contained in a file, one design per line. This file is useful if your design has many lower-level schematics.

Note: You can create a list file with the UNIX `ls` command. The `ls` command lists all the MGC components within a single directory, and the `sed` command strips the trailer from `.mgc_component.attr`. The result is directed to the list file.

```
ls *.mgc_component.attr |  
sed s/.mgc_component.attr//g > listfile
```

4. In the Enter Design Name field, enter the design name or the name of the file listing the designs to retarget.
5. In the Schematic Name field, enter the name of the schematic model.
The default is Schematic.
6. Select the **verbose** mode switch.
7. Leave the Check and Save Switch field set to its default setting, manual checking, to allow you to find Xilinx components that do not convert properly. Once you become familiar with Convert Design’s operation, you can select this field to have Convert Design automatically check and save the schematic.
8. In the From technology field, type the name of the device family from which you are converting. This field is case-insensitive.
9. In the To Technology field, type the name of the device family to which you are converting. This field is case-insensitive.
10. If you want the results of the conversion saved to a log file, type the name of the log file in the Log File Name field. The default is `log_file`.
11. Set a beep to sound for every un-matched symbol.
12. Click on **OK** to start the conversion.

Merging Design Files from Other Sources

You can enter part of your design in a 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 EDIF format. EDIF netlist files must be located in the working directory. Without the EDIF file, this portion of the design cannot be included; with it, the origin of the logic becomes irrelevant. To incorporate the EDIF file into your schematic, you must create a symbol for the file and place it on your schematic as you would any other component.

Simulation Models

Most Xilinx simulation models are built with 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 through the routed EDIF, Verilog, or VHDL file.

Analyzing Nets from the Schematic

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

You can probe nets in pld_quicksim by opening a schematic sheet and selecting a net. To trace the selected signal, follow these steps:

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 file to use in future pld_quicksim sessions. Refer to the pld_quicksim manuals from Mentor Graphics for detailed information on using the simulator and creating these files.

Setting Global Reset and 3-State Signals

The way you set Global Reset and 3-State signals depends on which part type you are using. The methods are described below.

FPGA Designs

Before you simulate an FPGA design, you must force the `//globalsetreset` (XC4000E designs) or the `//globalreset` (XC5200 designs) or the `//globalresetb` (XC3000 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** → `pld_quicksim` to enter the `pld_quicksim` simulator.
2. Select the **File** → **Open Sheet** menu item to display the Design Architect schematic.
3. Select the **Add Force** menu from the `pld_quicksim` Stimulus palette.
4. Fill in the dialog box with the `//globalsetreset` signal name, 25 for the first time, and 1 for the first value; n for the second time, and 0 for the second value.

It is recommended that you do not force signals at time 0. See Mentor's QuickSim user guide for details.

The reset width emulates a power-on reset at the beginning of simulation. `Globalsetreset` is now forced High at n ns. If you want to reset the flip-flops after n ns, toggle the `globalsetreset` Low and High for the necessary pulse width specified in *The Xilinx Programmable Logic Data Book*.

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

To set XC4000E/EX IOB flip-flops, follow these instructions:

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 `pld_quicksim` Stimulus palette.

3. Fill in the dialog box with the `//globalsetreset` signal name, 25 for the first time and 1 for the first value; n for the second time and 25 for the second value.

It is recommended that you do not force signals at time 0. See Mentor's QuickSim user guide for details.

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

XC4000E/EX parts have a global input state to make all output pins 3-state, which allows the isolation of the XC4000E/EX 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.

CPLD Designs

Before you simulate a XC7000 or XC9000 CPLD design, you must force the `//prld`, otherwise, the flip-flops do not function correctly.

1. Select your design directory icon in the Navigator window and select **Right Mouse Button** → **Open** → `pld_quicksim` to enter the `pld_quicksim` simulator.
2. Select the **File** → **Open Sheet** menu item to display the Design Architect schematic.
3. Select the **Add Force** menu from the `pld_quicksim` Stimulus palette.
4. Fill in the dialog box with the `//prld` signal name, 25 for the first time, and 1 for the first value; n for the second time, and 0 for the second value.

It is recommended that you do not force signals at time 0. See Mentor's QuickSim user guide for details.

The reset width emulates a power-on reset at the beginning of simulation. If you want to reset the flip-flops after n ns, toggle the `prld High` and `Low` for the necessary pulse width specified in *The Xilinx Programmable Logic Data Book*.

Manual Translation

You can access the programs required to simulate and implement your design through the graphical user interface of the Mentor Design Manager or through the UNIX command line.

The first half of this chapter discusses the program sequence for performing functional simulation, design implementation, and timing simulation from the UNIX command line for different types of designs. The second half describes the syntax of the individual programs.

This chapter contains the following sections:

- “**Functional Simulation**” section
- “**Design Implementation**” section
- “**Timing Simulation**” section
- “**Program Summary**” section

Functional Simulation

Pure Schematic Designs

1. Create a viewpoint using `pld_dve`:

```
pld_dve -s design_name technology [viewpoint_name]
```
2. Perform functional simulation with `pld_quicksim`:

```
pld_quicksim design_name[ /viewpoint_name]
```

Schematic Designs with XNF Elements

1. Create a symbol in `pld_da` for each XNF element in your design.

2. To the symbols, add the FILE property with the path name of the XNF file as the value.
3. Run `pld_men2edif` to convert the entire design into EDIF.
4. Run `pld_men2sim` on this EDIF file to create a design component that represents the entire design:

```
pld_edif2sim edif_file component_name technology -m -eddm [-sd dir]
```

Use `-sd` to search additional directories other than the one containing the source EDIF file to find supporting EDIF, NGO, or XNF files.

5. Perform functional simulation with `pld_quicksim`:

```
pld_quicksim design_name[ /viewpoint_name]
```

Schematic Designs with LogiBLOX Elements

Schematic designs with LogiBLOX elements already contain simulation models, so you only need to create a viewpoint, then simulate.

1. Create a viewpoint using `pld_dve`:

```
pld_dve -s design_name technology [viewpoint_name]
```

2. Perform functional simulation with `pld_quicksim`:

```
pld_quicksim design_name[ /viewpoint_name]
```

Mixed Schematic and VHDL with Schematic-on-Top Designs

You can simulate the design either before or after you synthesize the HDL module.

Before Synthesis

Follow these steps to simulate your design before you synthesize it:

1. Compile the VHDL module into a work library. If using Mentor version B.2 and up, use `-qhpro -syminfo` when compiling, otherwise Generate Symbol in the Design Analyzer will fail.
2. Create a symbol for the HDL module with `pld_da` using **File** → **Generate** → **Symbol**.

3. The Generate Symbol dialog box opens as shown in the “Generate Symbol Dialog Box” figure.
4. In the Generate Symbol dialog box, choose **Entity** as the source and specify the library logical name, entity name, and default architecture.
5. Instantiate the symbol on the schematic.
6. Create a viewpoint using pld_dve:


```
pld_dve -s design_name technology [viewpoint_name]
```
7. Run QuickHDL PRO to simulate the design by typing the following syntax:


```
qhpro [options] design_name
```

Figure 8-1 Generate Symbol Dialog Box

After Synthesis

To simulate your VHDL design after you synthesize it, follow these steps:

1. Synthesize the HDL module that is being included on the schematic, and create an EDIF file from that synthesis.
2. Create a symbol for the HDL module with `pld_da`.
3. If the synthesis output was an EDIF file, run `pld_edif2sim` to convert it to a Mentor EDDM single object:

```
pld_edif2sim edif_file symbol_component_name technology  
{-m|-s} -eddm [-sd dir1 ... -sd dirn]
```

Use `-m` if the synthesis was performed with a Mentor tool; use `-s` if the synthesis was performed with a Synopsys tool.

4. Perform functional simulation with `pld_quicksim`:

```
pld_quicksim design_name[ /viewpoint_name]
```

Where *design_name* is the EDDM design created by `pld_edif2sim`.

HDL-at-Top Designs

EDDM models must be inserted in the top-level HDL file.

1. Create a work library.
2. Perform the following steps for any schematic based components that need to be included in the top level VHDL:
 - a) Run `pld_dve -s` to create a viewpoint for each EDDM component.
 - b) Make sure the EDDM has an underlying symbol associated with it. If not create one using `pld_da` → **File** → **Generate Symbol**. Specify Schematics as the source in the dialog box.
 - c) Run `gen_arch` to create entity and architecture source files.
 - d) Instantiate this component into the top-level VHDL file.
3. Compile the VHDL source files with `qvhcom`:

```
qvhcom [options] design_name
```


See the Mentor documentation for a description of the available options.

4. Run QuickHDL PRO to simulate the design by typing the following syntax:

```
qhpro [options] design_name
```

For a description of the QuickHDL PRO options, see the Mentor Graphics documentation.

Pure HDL Designs

1. Create a working library.
2. Compile the HDL source files with qvhcom:

```
qvhcom [options] design_name
```

See the Mentor documentation for a description of the available options.

3. Simulate the design by running QuickHDL. Type the following syntax:

```
qhsim [options] [-lib_name] [primary [architecture [primary] ...]
```

Design Implementation

Schematic Designs (FPGA)

The procedure for implementing pure schematic designs, designs with XNF elements, designs with LogiBLOX elements, and mixed-model schematic-at-top designs is the same. Follow these steps:

1. Convert the EDDM design to EDIF format with pld_men2edif:

```
pld_men2edif design_name technology [viewpoint_name] [-b bus_delimiter]
```

2. Submit the design to NGDBuild, which reads a file in EDIF or XNF format, reduces all the components in the design to Xilinx primitives, runs a logical design rule check on the design, and writes an NGD file as output.

```
ngdbuild -p technology design_name
```

For example:

```
ngdbuild -p xc4000ex test -sd dir
```

3. Map the logic to the components in the FPGA by typing the following syntax:

```
map design_name.ngd -p partname
```

For example:

```
map -p 4000EXHQ240-3 test.ngd
```

4. Place and route the design:

```
par -w design_name.ncd design_name.ncd
```

The first file is created by the MAP utility, and PAR creates the other one.

For example:

```
par -w test.ncd test.ncd(writes out test.ncd created  
by map)
```

```
par -w test.ncd test_par.ncd(writes new file  
test_par.ncd)
```

5. Back-annotate the design:

```
ngdanno design_name.ncd design_name.ngm
```

6. Convert the design to an EDIF file:

```
ngd2edif -a -v mentor design_name.nga -w
```

7. Submit the design to pld_edif2tim, the Mentor EDIF netlist reader, which converts an EDIF netlist to a Mentor single-object EDDM file that can be submitted to pld_quicksim for timing simulation. Use this syntax:

```
pld_edif2tim design_name.edn
```

This step creates a design library, *design_lib*, containing the design on which you can perform timing simulation.

Schematic Designs (CPLD)

When using CPLDs, the procedure for implementing pure schematic designs, designs with XNF elements, and mixed-model schematic-at-top designs is the same. Follow these steps:

1. Convert the EDDM design to EDIF format with pld_men2edif:

```
p1d_men2edif design_name technology [viewpoint_name]
```

2. Submit the design to the CPLD fitter.

```
cpld -p partname design_name [-sd dir]
```

3. Convert the design to an EDIF file:

```
ngd2edif -a -v mentor design_name.nga -w
```

4. Submit the design to p1d_edif2tim, the Mentor EDIF netlist reader, which converts an EDIF netlist to a Mentor single-object EDDM file that can be submitted to p1d_quicksim for timing simulation. Use this syntax:

```
p1d_edif2tim design_name.edn
```

This step creates a design library, *design_lib*, containing the design on which you can perform timing simulation.

HDL-at-Top Designs

1. Synthesize the HDL modules in your design, and create an EDIF or XNF file from that synthesis.
2. Convert the EDIF or XNF file to an NGD file by using ngdbuild:

```
ngdbuild -p technology design_name
```

For example:

```
ngdbuild -p XC4000E test (where test is the root name  
for the EDIF or XNF file)
```

Note: Referenced Mentor EDDM models must have their corresponding EDIF files created and residing in the same directory where the top level EDIF or XNF file resides. If they reside in other directories, you must use the `-sd` option to specify additional directories to search for such files.

3. Map the logic to the components in the FPGA by typing the following syntax:

```
map design_name.ngd -p partname
```

For example:

```
map -p 4000EXHQ240-3 test.ngd
```

4. Place and route the design:

```
par -w design_name.ncd design_name.ncd
```

The first file is created by the MAP utility, and PAR creates the other one.

For example:

```
par -w test.ncd test.ncd (writes out test.ncd created by map)
```

```
par -w test.ncd test_par.ncd (writes new file test_par.ncd)
```

5. Back-annotate the design:

```
ngdanno design_name.ncd design_name.ngm
```

6. Convert the design to an EDIF file:

```
ngd2edif -a -v mentor design_name.nga -w
```

7. Submit the design to pld_edif2tim, the Mentor EDIF netlist reader, which converts an EDIF netlist to a Mentor single-object EDDM file that can be submitted to pld_quicksim for timing simulation. Use this syntax:

```
pld_edif2tim design_name.edn
```

This step creates a design library, *design_lib*, containing the design on which you can perform timing simulation.

Pure HDL Designs

1. Synthesize the HDL file, and create an EDIF or XNF file from that synthesis.
2. Convert the EDIF or XNF file to an NGD file by using ngdbuild:

```
ngdbuild -p technology design_name
```

For example:

```
ngdbuild -p XC4000E test (where test is the root name for the EDIF or XNF file)
```

Note: Referenced Mentor EDDM models must have their corresponding EDIF files created and residing in the same directory where the top level EDIF or XNF file resides.

3. Map the logic to the components in the FPGA by typing the following syntax:

```
map design_name.ngd -p partname
```

For example:

```
map -p 4000EXHQ240-3 test.ngd
```

4. Place and route the design:

```
par -w design_name.ncd design_name.ncd
```

The first file is created by the MAP utility, and PAR creates the other one.

For example:

```
par -w test.ncd test.ncd(writes out test.ncd created by map)
```

```
par -w test.ncd test_par.ncd(writes new file test_par.ncd)
```

5. Back-annotate the design:

```
ngdanno design_name.ncd design_name.ngm
```

6. Convert the design to an EDIF file:

```
ngd2edif -a -v mentor design_name.nga -w
```

7. Submit the design to `pld_edif2tim`, the Mentor EDIF netlist reader, which converts an EDIF netlist to a Mentor single-object EDDM file that can be submitted to `pld_quicksim` for timing simulation. Use this syntax:

```
pld_edif2tim design_name.edn
```

This step creates a design library, *design_lib*, containing the design on which you can perform timing simulation.

Timing Simulation

Schematic Designs

The procedure for performing timing simulation on pure schematic designs, designs with XNF elements, designs with LogiBLOX elements, and mixed-model schematic-at-top designs is the same. Follow these steps:

1. Use `pld_edif2tim` to create a Mentor EDDM model.

```
pld_edif2tim design_name.edn
```

1. Create a viewpoint using `pld_dve`:

```
p1d_dve -s design_lib/design technology [viewpoint_name]
```

2. Run p1d_quicksim to perform the timing simulation by using the following syntax:

```
p1d_quicksim -cp design_lib/design_name
```

This command brings up DVE for cross-probing.

For example:

```
p1d_quicksim -cp test_lib/test
```

3. Cross-probe between the original design and the new design.
4. Open the Viewpoint that was used to create the original design EDIF netlist.
5. Open the schematic sheet in p1d_dve.
6. Select the signals to trace in the p1d_dve schematic.

P1d_quicksim automatically creates a trace window and adds the selected signals to it. Use p1d_dve's schematic sheet window as if it were the sheet in the p1d_quicksim window.

Pure HDL Designs

You can create either an output EDIF file or output VHDL/Verilog file from the Xilinx Design Manager (or Xilinx core tool scripts).

EDIF Method

1. Submit the design to p1d_edif2tim, which converts an EDIF netlist to a Mentor single-object EDDM file that can be submitted to p1d_quicksim for timing simulation. Use this syntax:

```
p1d_edif2tim design_name.edn
```

This step creates a design library, *design_lib*, containing the design on which you can perform timing simulation.

2. Create a viewpoint with p1d_dve:

```
p1d_dve -s design_lib/design_name technology
```

3. Simulate the timing with p1d_quicksim:

```
p1d_quicksim design_lib/design_name
```

VHDL/Verilog Method

1. Compile the HDL source files with qvhcom:

```
qvhcom [options] design_name
```

See the Mentor documentation for a description of the available options.

2. Simulate the timing with QuickHDL:

```
qhsim options [-lib_name] [primary [architecture [primary]
...]
```

Program Summary

This section briefly describes the UNIX command-line syntax of the commands that activate the Mentor and Xilinx programs that you can use to process your designs manually. They are listed in alphabetical order.

CPLD

CPLD is a C-shell script for fitting into the XC7000 and XC9000 families. For a description of the CPLD command syntax and options, see the *CPLD Schematic Design Guide* or run the CPLD command with no parameters.

Dsgnmgr

Dsgnmgr, the Xilinx Design Manager, is Xilinx's design implementation tool.

The dsgnmgr syntax can take the following three forms:

```
dsgnmgr
```

```
dsgnmgr project
```

```
dsgnmgr -design design.edif
```

When you use the first form of the syntax, the Design Manager appears with no project loaded. A project in this context means a Xilinx project.

When you use the second form of the syntax, the Design Manager appears but with the specified project loaded or opened. The project is a fully specified file name with a .prj extension. It is a file created by

the Design Manager and contains the project information for a Xilinx project.

When you use the third form of the syntax, the Design Manager finds the design. A design in this context is a netlist file such as an EDIF file. If the design does not already have a Xilinx project associated with it, the Design Manager creates a project and appears with this project loaded. If the design does already have a Xilinx project associated with it, the Design Manager appears with that project loaded.

EDIF2NGD

Edif2ngd converts an EDIF 2 0 0 netlist to a Xilinx NGO file. The EDIF file includes the hierarchy of the input schematic. The output NGO file is a binary database describing the design in terms of the components and hierarchy specified in the input design file.

For a description of the edif2ngd syntax and options, see the *Development System Reference Guide*.

Editor

The Notepad editor is a full-featured, window-based text editor. It is only available in the graphical user interface of the Mentor tools.

Gen_Arch

Gen_Arch creates VHDL entity and architecture from a Mentor (EDDM) component.

For a description of the Gen_Arch syntax and options, see the Mentor Graphics documentation.

MAP

MAP is a Xilinx tool that maps the logic to the components in an FPGA design.

For a description of the MAP syntax and options, see the *Development System Reference Guide*.

NGDAnno

NGDAnno is Xilinx's back-annotation utility.

For a description of the NGDAnno syntax and options, see the *Development System Reference Guide*.

NGDBuild

NGDBuild reads a file in EDIF or XNF format, reduces all the components in the design to Xilinx primitives, runs a logical design rule check on the design, and writes an NGD file as output.

For a description of the NGDBuild syntax and options, see the *Development System Reference Guide*.

NGD2EDIF

Ngd2edif converts a Xilinx NGD or NGA file to an EDIF 2 0 0 netlist.

For a description of the ngd2edif syntax and options, see the *Development System Reference Guide*.

PAR

PAR is Xilinx's place and route tool.

For a description of the PAR syntax and options, see the *Development System Reference Guide*.

Pld_da

Pld_da is Design Architect, a schematic editor configured for Xilinx designs. For a description of Design Architect, see the *Mentor Graphics Design Architect Users Manual*.

Pld_dve

Pld_dve creates a simulation or custom viewpoint for a Xilinx design.

The pld_dve syntax is the following:

```
pld_dve [-s] design_name technology [viewpoint_name]
```

- -s creates a simulation viewpoint for pld_quicksim (chip-level/board-level functional/timing). It is optional. If you do not use -s but specify a viewpoint name, pld_dve opens in the interactive mode and opens the specified viewpoint.
- *design_name* is the name of your Mentor design component.

- *technology* specifies the PLD architecture.
- *viewpoint_name* specifies the name of the design viewpoint to generate. This is optional; *pld_dve* does not perform any customization on this viewpoint if *-s* is not specified.

When *pld_dve* creates a simulation viewpoint—that is, when you use the *-s* option—and if the viewpoint contains COMP or FILE primitives, *pld_dve* removes these primitives, then creates a viewpoint that can be submitted to *pld_quicksim*.

Pld_edif2sim

Pld_edif2sim is a utility that converts a Mentor, Synopsys, or any other Xilinx compatible EDIF file into a Mentor EDDM single object, VHDL netlist, Verilog netlist, or NGO file.

The *pld_edif2sim* syntax is the following:

```
pld_edif2sim edif_file symbol_component_name | output_file_name
technology {-s|-o|-m} {-eddm|-vhdl|-verilog|-ngo}
{-hier|-flat} {-ignore_unexpanded} [-sd dir1 ... -sd
dirn] [-help]
```

- *edif_file* is the name of the EDIF file from Mentor, Synopsys, or Data I/O.
- *symbol_component_name* is the name of the component. This is used for the *-eddm* option.
- *output_file_name* is the name of the output VHDL or Verilog. This is used for the *-vhdl* or *-verilog* options.
- *technology* specifies the PLD architecture.
- *-ngo* specifies that *pld_edif2sim* should produce a *design_name.ngo* file only.
- *-s* indicates that the EDIF file is a Synopsys file.
- *-o* indicates that the EDIF file is any third party vendor's EDIF that is compatible with Xilinx.
- *-m* indicates that the EDIF file is a Mentor file.
- *-eddm* specifies that the EDIF file be converted to Mentor's EDDM single object.
- *-vhdl* specifies that the EDIF file be converted to a VHDL file.

- `-verilog` specifies that the EDIF file be converted to a Verilog file.
- `-sd` specifies additional directories to search to find any supporting EDIF, XNF, or NGO files.
- `-hier` specifies that the VHDL/Verilog netlist is hierarchical.
- `-flat` specifies that the VHDL/Verilog netlist is flat (default is flat).
- `-ignore_unexpanded` specifies that if there are any unknown primitives in the design, `pld_edif2sim` does not exit with an error status; instead, it ignores this condition and goes on. By default, it exits with an error message.
- `-help` allows you to obtain more information on `pld_edif2sim` and its options. It is optional.

Pld_edif2tim

`Pld_edif2tim` is the Mentor EDIF netlist reader, which converts an EDIF netlist to a Mentor single-object EDDM file that can be submitted to `pld_quicksim` for timing simulation.

The `pld_edif2tim` syntax is the following:

```
pld_edif2tim edif_file [-r] [-help]
```

- `edif_file` is the name of the EDIF file.
- `-r` specifies that if `design_lib` already exists, it will be replaced.
- `-help` allows you to obtain more information on `pld_edif2tim` and its options. It is optional.

Pld_men2edif

`Pld_men2edif` is the Mentor EDIF netlist writer, which creates a hierarchical EDIF netlist from a Mentor schematic design.

The `pld_men2edif` syntax is the following:

```
pld_men2edif design_name technology [viewpoint_name]
[-b 'delimiter'] -circular [-help]
```

- `design_name` is the name of your Mentor design component.
- `technology` specifies the PLD architecture.
- `viewpoint_name` specifies the name of the design viewpoint to use. It is optional. If a viewpoint does not exist, `pld_men2edif` will

create one. If you do not specify the viewpoint, it will use the viewpoint called default.

- `-circular` overcomes the forward referencing problem that occurs if a primitive in one library is referenced in another library before its parent library is defined in EDIF. In this case the EDIF reader fails to process the EDIF file. The `-circular` switch prevents this problem.
- `-b 'delimiter'` specifies the bus dimension separator style as an angle bracket, square bracket or paren.

delimiter is one of the following: Angle | Square | Paren

The `-b` option instructs the EDIF writer to convert the bus delimiters into the specified delimiter. If `-b` is not specified, `'()` will be used for bus delimiters by default.

- `-help` allows you to obtain more information on `pld_men2edif` and its options. It is optional.

Pld_quicksim

`Pld_quicksim` is an interactive logic simulator that performs functional or timing simulation on your designs.

The `pld_quicksim` syntax is the following:

```
pld_quicksim [-cp] design_name[ /viewpoint_name]
```

- `-cp` ensures that cross-probing is performed. It is optional. If you specify this option, QuickSim invokes DVE to allow viewing the front-end schematic for cross-probing. You must then open the viewpoint on the original design that was used to create the EDIF netlist.
- *design_name* is the name of your Mentor design directory.
- *viewpoint_name* specifies the name of the design viewpoint to use. It is optional. If you specify a viewpoint name, it must be preceded with a slash and appended to the design name, as in the following example:

```
pld_quicksim test/myvpt
```

For a description of the other options available in `pld_quicksim`, see the Mentor Graphics *QuickSim Users and Reference Manuals*.

To enable cross-probing between front-end and back-end designs in timing simulations, specify `-cp`. In this case, the syntax is the following:

```
p1d_quick5im -cp test_lib/test
```

P1d_xnf2sim

P1d_xnf2sim is a utility that converts an XNF file to a Mentor EDDM single object, VHDL netlist, or Verilog netlist.

The p1d_xnf2sim syntax is the following:

```
p1d_xnf2sim top-level_xnf_file [-list listfile]
symbol_component_name | output_file_name technology
{-ignore_unexpanded} [-s] {-eddm|-vhdl|-verilog}
{-hier|-flat} [-sd dir1 ... -sd dirn] [-help]
```

- *top-level_xnf_file* is the top-level XNF file.
- `-list listfile` allows you to list all the related XNF files to be converted. It is optional. If you do not specify `-list`, all XNF files located in the directory in which the top-level XNF file resides are used as referenced by the top-level XNF file.
- *symbol_component_name* is the name of the Mentor component for which a simulation model is to be created.
- *output_file_name* is the name of output VHDL or Verilog (for `-vhdl` or `-verilog` option)
- *technology* specifies the PLD architecture.
- `-s` indicates that the XNF file is a Synopsys file. It is optional.
- `-eddm` specifies that the XNF file be converted to an EDDM single object.
- `-vhdl` specifies that the XNF file be converted to a VHDL file.
- `-verilog` specifies that the XNF file be converted to a Verilog file.
- `-hier` specifies that VHDL/Verilog is hierarchical.
- `-flat` specifies that VHDL/Verilog is flat (This is the default).
- `-ignore_unexpanded` specifies that if there are any unknown primitives in the design, p1d_edif2sim does not exit with an error status; instead, it ignores this condition and goes on. By default, it exits with an error message.

- `-sd` specifies additional directories to search to find any supporting EDIF, XNF, or NGO files.

XNF file(s) submitted to `pld_xnf2sim` must represent the entire design, including the top-level IO ports (EXT statements). Feeding an XNF file that only represents one part of a design (with no IO pads) results in an invalid simulation model. You can use the following procedure to run functional simulation on a schematic design that consists of a partial XNF:

1. Create symbols representing the XNF files.
2. Add the FILE property with the value equal to the pathname of the XNF file.
3. Instantiate these symbols on your schematic.
4. Create an EDIF file with `pld_men2edif` (using the top-level schematic).
5. Feed this EDIF file to `pld_edif2sim` to create an EDDM model.
6. Simulate this EDDM model with `pld_quicksim`.

QuickHDL

QuickHDL (`qhsim`), is Mentor's simulator for behavioral VHDL, Verilog, VHDL-based, and Verilog-based gate-level designs composed of Unified Libraries or SIMPRIM elements.

The QuickHDL syntax is the following:

```
qhsim options [-lib_name] [primary [architecture  
[primary] ...]
```

For a description of the QuickHDL options, see the Mentor Graphics documentation.

QuickHDL PRO

QuickHDL PRO (`qhpro`) is Mentor's simulator for mixed-model schematic, VHDL, and Verilog designs. It can invoke QuickHDL to simulate HDL-based elements, or QuickSim to simulate gate-level schematics.

The QuickHDL PRO syntax is the following:

```
qhpro options design_name
```

For a description of the QuickHDL PRO options, see the Mentor Graphics documentation.

QuickPath

QuickPath performs a static and slack timing analysis on designs. For a description of the QuickPath syntax and options, see the Mentor Graphics documentation.

Qvhcom

Qvhcom compiles the VHDL to be able to run QuickHDL (qhsim) simulator.

```
qvhcom [options] design_name
```

See the Mentor documentation for a description of the available options.

Qvlcom

Qvlcom compiles the Verilog files to be able to run QuickHDL (qhsim) simulator.

```
qvlcom [options] design_name
```

See the Mentor documentation for a description of the available options.

SysArch

SysArch is the System Architect, which creates system-level designs. For a description of the SysArch syntax and options, see the Mentor Graphics documentation.

Pld_sg

Pld_sg invokes the Mentor schematic generator (SG), which creates a schematic from an EDDM model. You must have a Mentor schematic generator license in order to use this tool. Usage is as follows:

```
pld_sg [options] [viewpoint_path]
```

See the Mentor documentation for a description of the available options.

Schematic Design Tutorial

This chapter contains the following sections:

- “Introduction” section
- “Required Background Knowledge” section
- “Design Flow” section
- “Software Installation” section
- “Starting the Design Manager” section
- “Copying the Tutorial Files” section
- “Starting Design Architect” section
- “Targeting the Design for the XC9000 Family” section
- “Completing the Calc Design” section
- “Controlling FPGA/CPLD Layout from the Schematic” section
- “Modifying the Design for Non-XC4000E/EX Devices” section
- “Using LogiBLOX” section
- “Other Special Components” section
- “Using a Constraints File” section
- “Performing Functional Simulation” section
- “Using Pld_men2edif” section
- “Using the Xilinx Design Manager” section
- “Performing Timing Simulation” section
- “Examining Routed Designs with EPIC” section
- “Verifying the Design Using a Demonstration Board” section

- “Making Incremental Design Changes” section
- “Command Summaries” section
- “Further Reading” section

Introduction

This chapter guides you through a typical field-programmable gate array (FPGA) and complex programmable logic device (CPLD) design procedure from schematic entry to completion of a functioning device. It uses a design called Calc, a 4-bit processor with a stack. In the first part of the tutorial, you use the Design Architect, the Mentor Graphics design entry tool, to create the schematics and symbols for the Calc design. Next you use `pld_quicksim`, the Mentor Graphics simulator, to perform a functional simulation on it. In the third step, you use the Xilinx Design Manager to implement the design. Finally, you verify the design’s timing by again using `pld_quicksim`. The simple design example used in this tutorial demonstrates many system features that you can apply to more complex FPGA and CPLD designs.

Note: Although this tutorial describes creating and processing FPGA designs, you can apply most of the steps to CPLD designs.

This tutorial includes instructions on the following:

- Installing the tutorial files
- Using Mentor Graphics Design Manager
- Targeting the tutorial design (Calc) for an XC4000E or an XC9000 device
- Using Design Architect
- Completing the ALU block in the Calc design
- Adding the STARTUP block to tie signals to the global reset
- Adding device information in the Calc design
- Exploring Xilinx library elements
- Exploring the XC4000E oscillator
- Controlling device layout from the schematic
- Editing the Calc design for a non-XC4000E/EX device

- Performing functional simulation on the Calc design in pld_quicksim
- Converting the design to an EDIF file using pld_men2edif
- Implementing the design using pld_dsgnmgr
- Configuring the Xilinx Design Manager/Flow Engine
- Performing timing simulation on the routed Calc design in pld_quicksim
- Examining routed designs with the Editor for Programmable ICs (EPIC)
- Verifying the 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
- 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.

Design Flow

See the “**Design Flows**” section of the “**Introduction**” chapter for the design flow involved in using the Mentor Graphics interface. That chapter also describes the general steps for creating a design using the Mentor interface.

This tutorial describes an incremental design methodology. In incremental design, you process the design, make a small change to the design, and process the design again. You use place and route information from the previous design processing cycle to constrain subsequent cycles of the same design. When you use this method, 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.

You can target the tutorial design for an XC4000E or XC9000 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.

This tutorial uses the following conventions to refer to the various device families:

- **XC3000 family**—includes XC3000, XC3000A, XC3000L, XC3100, and XC3100A devices
- **XC4000 family**—includes XC4000, XC4000E, XC4000EX, XC4000L, and XC4000XL devices
- **XC5200 family**—includes XC5200 devices
- **XC9000 family**—includes XC9500 and XC9500F devices

Software Installation

Required Software

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

- Mentor Graphics Version B.1 or later, including Mentor Design Manger, Design Architect, QuickSim, QuickPath, as well as the programs needed to read and write EDIF netlists (ENRead and ENWrite), which require special licensing
- Xilinx/Mentor Graphics Interface Version M1
- Xilinx Development System Version M1

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:
 - Xilinx Development System Version M1
 - Xilinx/Mentor Graphics Interface Version M1
 - Mentor Graphics Version B.1 or later, including Mentor Design Manger, Design Architect, QuickSim, QuickPath, as well as the programs needed to read and write EDIF netlists (ENRead and ENWrite), which require special licensing
3. Verify the installation, using the **“Configuring Your System” section of the “Getting Started” chapter** as a guide.
4. Add a reference to \$XILINX_TUTORIAL to your MGC_LOCATION_MAP file.

Every symbol and schematic in your design contains references which indicate where design objects reside on your disk or network. 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 in their path references. \$XILINX_TUTORIAL must be defined in the file pointed to by \$MGC_LOCATION_MAP. For example, the design object seg7dec in the \$XILINX/mentor/tutorial/calc_sch directory uses the path reference \$XILINX_TUTORIAL/calc_sch/seg7dec 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 you make a query to determine where the design object “\$XILINX_TUTORIAL/calc_sch/stack” is located, the Mentor Graphics tools use this definition to determine that stack is at /home/bclinton/mentor/xtutorial/calc_sch/stack.

With this definition added to the location map as defined in the **“Getting Started” chapter**, the complete location-map file should, at a minimum, look like:

```
MGC_LOCATION_MAP_1
(empty line)
$MGC_GENLIB
(empty line)
```

```
$LCA  
(empty line)  
$SIMPRIMS  
(empty line)  
$XILINX_TUTORIAL  
/home/bclinton/mentor/xtutorial
```

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

Installing the Tutorial

You can optionally install the tutorial files when you install the Xilinx/Mentor Graphics interface software. If you have already installed the software, but are not sure whether you specified tutorial installation, check your software installation for a `$XILINX/mentor/tutorial` directory. This directory contains the tutorial files.

Standard Directory Structure

When you create a design object 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_name.mgc_component.attr`, that resides at the same level as the directory which has the name. For example, if you create a schematic named `calc`, a `calc` directory is created, and at the same level the file, `calc.mgc_component.attr`, is created. The `calc` directory contains 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 `$XILINX/mentor/tutorial` directory is created; design object directories are created; and the tutorial files needed to complete the design are copied to the `calc_sch` directory. Some of the files you need to complete the tutorial design are not copied, because you create these files in the tutorial. However, solutions directories with all input and output files are provided. They are located in the

\$XILINX/mentor/tutorial directory and are listed in the following table.

Table 9-1 Tutorial Design Directories

Directory	Description
calc_sch	Schematic (Design Architect) tutorial directory
calc_4ke	Schematic solution directory for XC4003E-PC84
calc_9k	Schematic solution directory for XC95108-PC84
calc_sot	Schematic-on-top tutorial directory (uses XC4003E)

The solution directories contain the design files for the completed tutorial, including schematics, intermediate files, and the bitstream file. Different intermediate files are created for different device families. Do not overwrite any files in the solutions directories.

The calc_sch directory contains the incomplete copy of the tutorial design. The installation program copies a few intermediate files to the calc_sch tutorial directory, and you create the remaining files when you perform the tutorial. As described in a later step, you copy the calc_sch directory to another area and perform the tutorial in this new area. The following table lists and describes the directories and files in the calc_4ke solution directory.

Table 9-2 Tutorial Directories/Files in the Calc_4ke 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
muxblk2	Design component for arithmetic function in ALU
andblk2	Design component for arithmetic function in ALU
clockgen	System-clock generator
orblk2	Design component for arithmetic function in ALU

Table 9-2 Tutorial Directories/Files in the Calc_4ke Directory

Directory or File Name	Description
xorblk2	Design component for arithmetic function in ALU
muxblk5	Design component for multiplexer arithmetic outputs in ALU
muxblk2a	Design component for multiplexer operator function in control
stack	Design component for stack
seg7dec	Design component for 7-segment decoder
debounce	Design component for debounce circuit
calc.edif	EDIF netlist files created by pld_men2edif
pld_men2edif.log	pld_men2edif log file
calc.ngo	Native Generic Object created by EDIF2NGD
calc_4ke.ucf	User Constraints File
calc.bld	Design database report generated by NGDBUILD
calc.ngd	Native Generic Design created by NGDBUILD
calc.mrp	Mapping report generated by MAP
calc.pcf	Physical Constraints File created by MAP
calc_map.ncd	Native Circuit Description created by MAP
calc.par	Place-and-Route report generated by PAR
calc.pad	Pinout description generated by PAR
calc.ncd	Routed NCD file created by PAR
calc.twr	Timing report generated by Trace (TRCE)
calc.bit	Configuration bitstream created by BITGEN
calc.edn	Timing-model EDIF netlist created by NGD2EDIF
calc_lib/calc	QuickSim timing simulation model created by pld_edif2tim
pld_edif2tim.log	pld_edif2tim log file

In addition to the files listed above, there is a file called *file-name.mgc_component.attr* 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:

```
p1d_dmgr
```

The Design Manager Window appears as shown in the following figure.

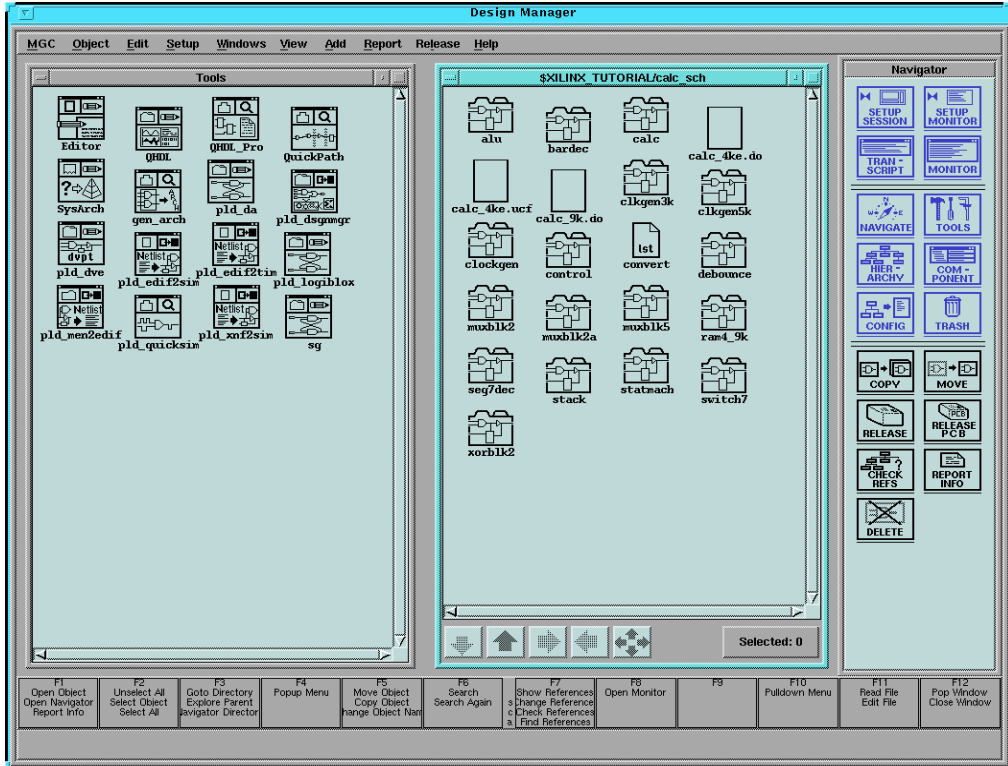


Figure 9-1 Mentor Design Manager Window

The Design Manager window contains the following three sub-windows:

- Tools Window
- Navigator Window
- 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 tutorial. A description of these programs is given in the **“Features” section of the “Introduction” chapter.**

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 five buttons located at the bottom of the window, three of which are most used. 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 rightmost 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

Mentor Graphics design objects contain absolute directory path references. Since many of these paths are self-referencing, using the `cp` or `mv` command in Unix to copy or move these design objects to new directories can break those references. The Mentor Graphics Design Architect allows design objects to be copied or moved across directories by adjusting path references within each design object as it is relocated.

To demonstrate the Copy operation in Design Manager, perform the following steps:

1. In the Navigator window, move to the directory where the tutorial files were installed.
2. Select the `calc_sch` directory.
3. To see the references in this design, choose **Right Mouse Button** → **Report** → **Show References** → **For Design**.

A List of Unique References underneath the `calc_sot` directory is displayed. An example reference item might be:

```
$XILINX_TUTORIAL/calc_sot/alu/alu:mgc_symbol[6]
```

This indicates that an ALU symbol (version 6 under Mentor Graphics' versioning system) is referenced by the path `$XILINX_TUTORIAL/calc_sot/alu/alu`.

All references in the design should contain either `$LCA` or `$XILINX_TUTORIAL`.

4. Close the List of Unique References window.
5. With the `calc_sch` directory selected in the Navigator window, choose **Right Mouse Button** → **Edit** → **Copy**.

A dialog box appears.

6. 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_sch`. Click OK.
7. Use the Navigator to change directories to the location of the working copy of `calc_sch`. In the example above, you would click

the “four-arrow” button at the bottom of the Navigator window, then type `/home/dum/tutor/mentor` in the dialog box.

8. Select the `calc_sch` directory.
9. As before, choose **Right Mouse Button** → **Report** → **Show References** → **For Design**. The example reference above with the ALU symbol appears:

```
/home/dum/tutor/mentor/calc_sch/alu/alu:mgc_symbol[6]
```

10. Close the List of Unique References window.
11. Modify your `MGC_LOCATION_MAP` file so that the `$XILINX_TUTORIAL` variable points to the directory where the copy of `calc_sch` is located. In the example above, change the `$XILINX_TUTORIAL` section of the file so that it reads:

```
$XILINX_TUTORIAL  
/home/dum/tutor/mentor
```

12. Read the newly modified location map into Design Architect by selecting **MGC** → **Location Map** → **Read Map** from the menu bar.
13. In the dialog box, type `$MGC_LOCATION_MAP`, then click **OK**.

The `$XILINX_TUTORIAL` soft name now points to the new tutorial area. However, references in the `calc_sch` directory use `/home/dum/tutor/mentor` instead of its new equivalent, `$XILINX_TUTORIAL`. While this is legal, it is best in Mentor to use soft names wherever possible.

14. To convert the hard name back into a soft name, select the `calc_sch` directory and choose **Right Mouse Button** → **Edit** → **Change** → **References**.
15. In the Change References dialog box, enter for From: `/home/dum/tutor/mentor` (or whatever directory is applicable to your case). For To, enter `$XILINX_TUTORIAL`.

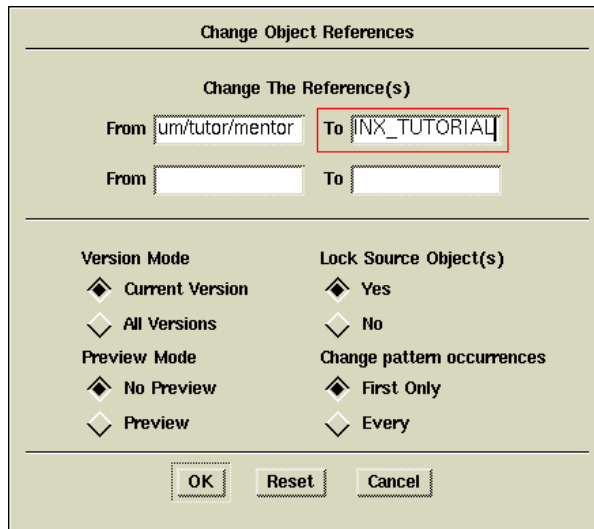


Figure 9-2 Change References Dialog Box

16. Click **OK**.

The Change References process begins.

17. After the process is finished, you can do another Show References operation to verify that all references have been changed properly.

Note: You can copy or move a design object without rewriting path references by selecting **Options** → **Convert References?** **No** from the Copy or Move dialog box.

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/calculator** in the Directory field of the dialog box, then select **OK** or press return.

This sets the working directory to the directory where you work on the tutorial.

3. Select the \$XILINX_TUTORIAL/calc_sch/calc design object in the Navigator window.
4. Select **Right Mouse Button** → **Open** → **p1d_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.

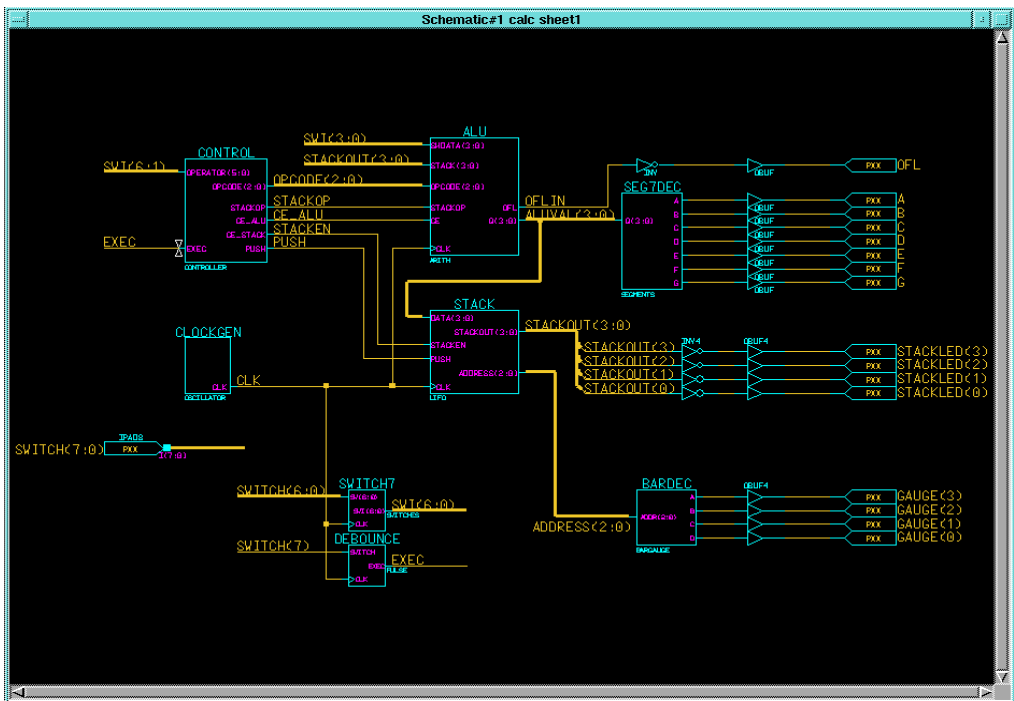


Figure 9-3 Top-Level Schematic for Calc

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. Design Architect converts the shape you draw to a number string to determine which command to execute. The number is determined as shown in the following figure:

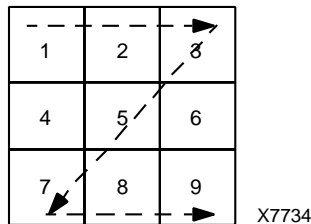


Figure 9-4 Using Strokes, Example of “Z” stroke (1235789)

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 draw the shape of a question mark “?” to display the stroke help screen. When applicable, this tutorial uses strokes and describes them using the numbering system shown in the [“Using Strokes, Example of “Z” stroke \(1235789\)” figure](#).

Right Mouse Button

You can use the right mouse button to display different menus depending on the object(s) selected on the schematic sheet. For

example, if a net is selected when you press the right mouse button, the Net menu appears. You can access other menus, regardless of what is selected, by using the “Other Menus” selection that appears at the top of each menu.

Using the Function Keys

You can use the keyboard function keys to execute many Design Architect commands. The boxes at the bottom of the Design Architect window each contain up to four commands which you execute as follows:

- To execute the top command, press the associated Function key.
- To execute the middle command, press the associated Function key while holding down the Shift key.
- To execute the third command, press the associated Function key while holding down the Control key.
- To execute the bottom command, press the associated Function key while holding down the Alternate 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, you can open a schematic sheet by typing **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 using either 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. You can also perform this function with the stroke 12369, which looks like an upside-down “L”.

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 these commands with the strokes 159 and 951, respectively. You can also zoom the schematic in or out with the menu bar commands **View** → **Zoom In** and **View** → **Zoom Out** or the strokes 357 and 753.

Targeting the Design for the XC9000 Family

The incomplete calc_sch design is configured for an XC4003E-PC84 part. If you want to target a demonstration board with this device, go to the **“Completing the Calc Design”** section. If you are targeting the

tutorial design for an XC95108-PC84 (no demonstration board available) or other device family, you must convert the design to reference the XC9000 library instead of the XC4000E library.

The procedure provided below allows you to change every Xilinx component in the Calc design from the XC4000E library to the XC9000 library. Since the designs were created using the Unified Libraries, the parts in the XC4000E and XC9000 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. This example shows a situation where this may happen.

Note: Although an XC4000E-to-XC9000 conversion is shown here, this procedure may be used to retarget from any family to any other family.

To retarget the Calc design to the XC9000 family:

1. Close the Calc schematic window by selecting **C**lose from the window's control menu. This is the menu accessed by clicking on the button in the upper left-hand corner of the window.
2. In the gray desktop area, choose **R**ight **M**ouse **B**utton → **C**onvert **D**esign.
3. A dialog box appears as shown in the figure below. Fill in the fields as shown and click **O**K.

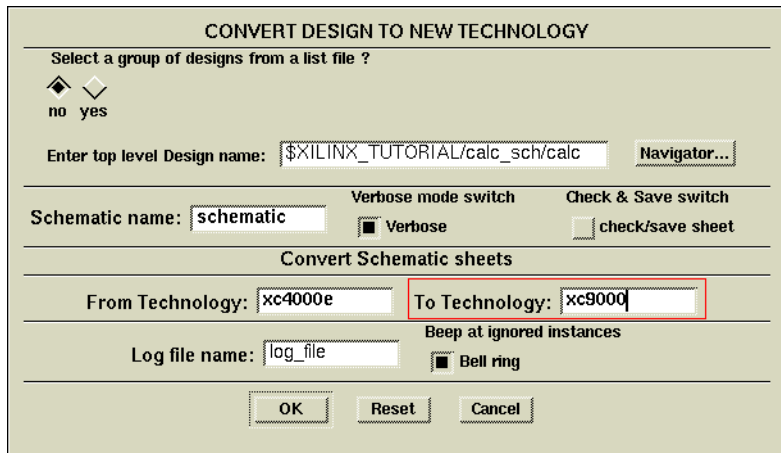


Figure 9-5 Convert Design Dialog Box

Convert Design begins by displaying a Hierarchy Window as it is taking account of the design hierarchy. After this, the Calc schematic and all lower-level design schematics are displayed while Design Architect is retargeting design components.

4. After this process is completed, perform the following for each schematic:
 - a) Select **C**heck → **S**heet from the menu bar. A window appears containing the results of the design rule check.
 - b) After reviewing the contents of this window, close it and reselect the schematic window.
 - c) Select **F**ile → **S**ave from the menu bar to save the schematic.
 - d) Close the schematic window and repeat the process for the next schematic.

Note: Keep the top-level Calc design open, since you need it in the next section. You can also save yourself the step of checking and saving each design sheet by setting the Check & Save switch to Yes in the Convert Design dialog box.

Warning: Although turning the Check & Save switch on makes the Convert Design process more automatic, it is more dangerous since it prevents you from inspecting a converted design before saving it.

Use this setting with caution, and turn it on only when you are certain you wish to overwrite your original design.

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 your work as follows:

1. Select **Check** → **Sheet** from the menu bar.
A window appears containing the results of the design rule check.
2. After reviewing the contents of this window, close it and reselect the schematic window.

Warning: It is important to check your design first before saving it.

3. Select **File** → **Save** from the menu bar to save the design.
4. Before proceeding to the next part of this tutorial, close (quit) the Calc schematic window.
5. If a dialog box appears asking if you want to save any changes, choose **NO**.

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 add the FD4RE symbol from the Unified Libraries to the ALU block. After the ALU block is finished, you add the STARTUP block to the top-level Calc schematic to tie the device's global reset network to a device pin. To complete design entry, you

add a CONFIG block, which lists a set of instructions that dictate how the core tools should process the design.

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. A count of the items in the stack is displayed as a “gauge” on another bar LED.

The design consists of the following 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 synchronous RAM in the XC4000E design. You can substitute the RAM4_9K module, which uses flip-flops, in place of the RAM16X4S macro in the STACK schematic to implement the stack in an XC9000 or other non-XC4000E device.

Note: If RAM4_9K is used in a non-XC9000 device, it must be retargeted using Convert Design.

- **DEBOUNCE**—This circuit debounces the “execute” switch, providing a one-shot output.
- **SEG7DEC**—This block decodes the output of the ALU for display on the 7-segment decoder.
- **CLOCKGEN**—This block uses an internal oscillator circuit in XC4000E devices to generate the clock signal. In XC9000 designs, it is replaced by an input pad and a clock buffer.

Note: The XC3000 and XC5200 FPGA families also have on-board oscillators. See the CLKGEN3K and CLKGEN5K components included in the calc_sch directory to see how the oscillators in these families are used.

- **BARDEC**—This block shows how many items are on the stack on a “gauge” of four LEDs.
- **SWITCH7**—This block is a user-defined module consisting of seven input flip-flops used to latch the switch data.

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_sch/andblk2` in the Component Name box.
3. 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 in the following figure and then select **OK**.

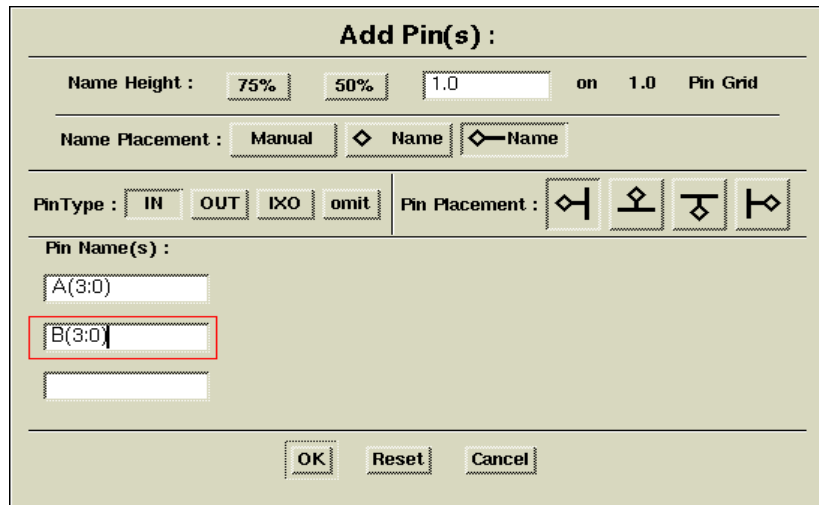


Figure 9-6 Add Pin(s) Dialog Box for A(3:0) and B(3:0)

A small crosshair appears under the cursor, and a rectangular box appears stating that the first pin, A(3:0), is to be placed.

3. Place pin B(3:0) 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.

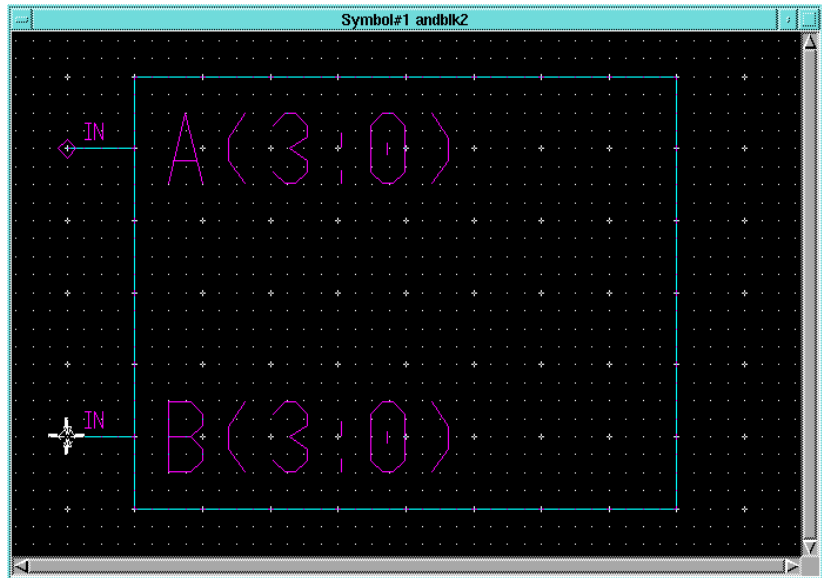


Figure 9-7 Adding Pins A(3:0) and B(3:0)

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

The image shows a dialog box titled "Add Pin(s) :". It has several sections:

- Name Height :** Three input fields with values "75%", "50%", and "1.0". To the right are the labels "on", "1.0", and "Pin Grid".
- Name Placement :** A dropdown menu set to "Manual", followed by a diamond icon, the text "Name", another diamond icon, and the text "Name".
- PinType :** Four buttons labeled "IN", "OUT", "IXO", and "omit".
- Pin Placement :** Four icons representing different pin symbols: a diamond with a vertical line, a diamond with a horizontal line, a diamond with a vertical line and a horizontal line, and a diamond with a horizontal line and a vertical line.
- Pin Name(s) :** A text input field containing "Q(3:0)" which is highlighted with a red rectangle. Below it is an empty text input field.
- Buttons:** "OK", "Reset", and "Cancel" buttons at the bottom.

Figure 9-8 Add Pin(s) Dialog Box for Q(3:0)

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

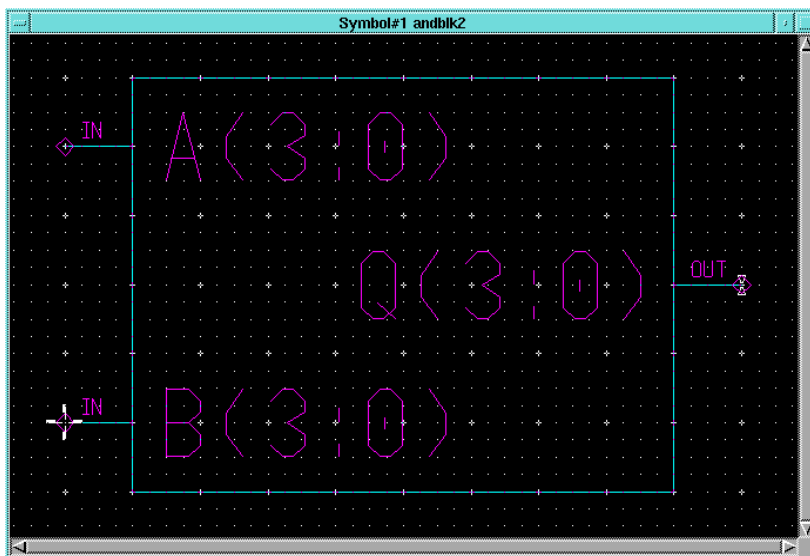


Figure 9-9 Adding Pin Q(3:0)

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.

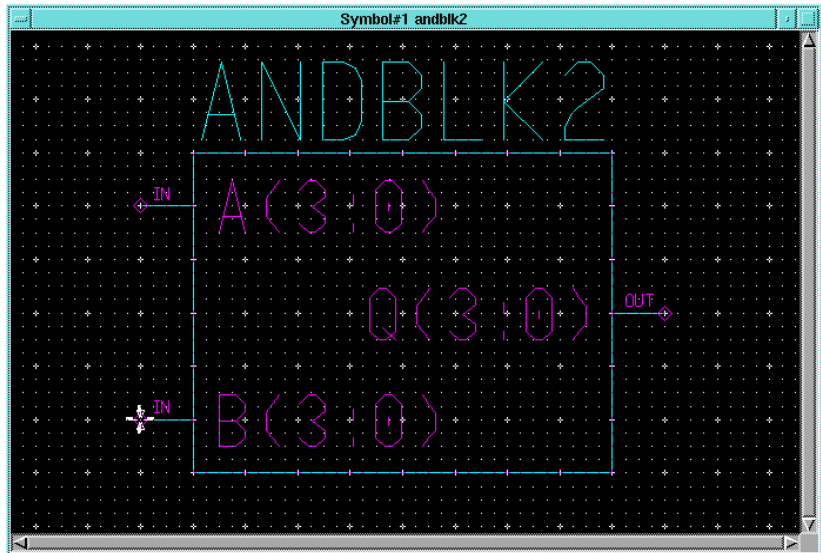
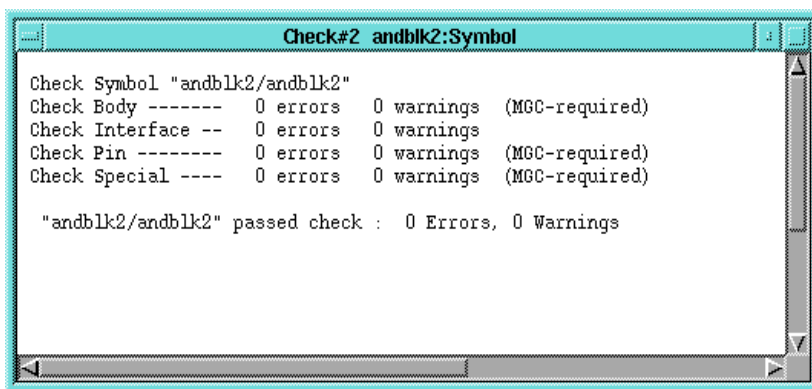


Figure 9-10 Completed ANDBLK2 Symbol

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.



```
Check#2 andblk2: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 9-11 Output from Check

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.

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 as described below.

1. Move the cursor above the ANDBLK2 text.
2. Press the F7 key while holding down the shift key to select the Change Text Value command.

3. In the small dialog box that appears in the lower left corner, type **ORBLK2** in the New Text field, then select **OK**.

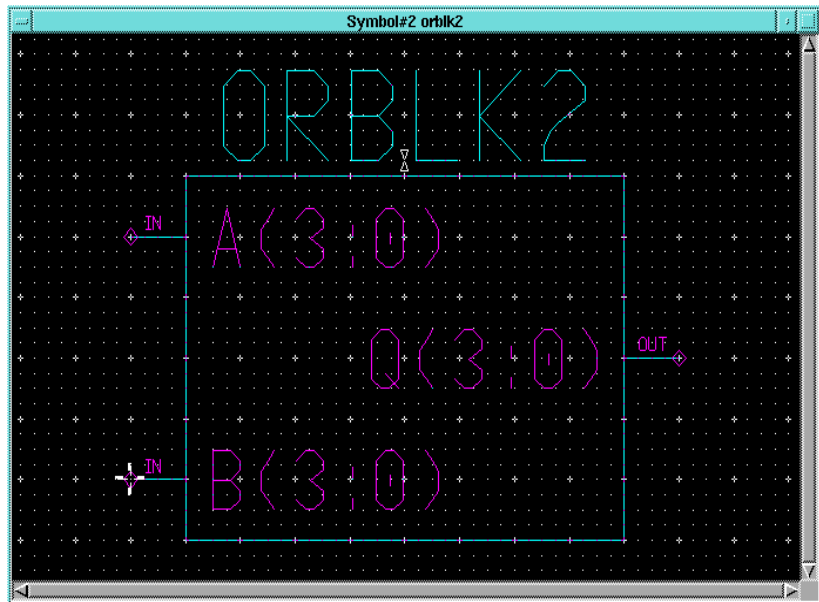


Figure 9-12 Completed ORBLK2 Symbol

4. If necessary, use the cursor and F7 key to move and center the text, as described earlier.
5. 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.
6. 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.
7. 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.

8. Enter `$XILINX_TUTORIAL/calc_sch/orblk2` in the component name field and enter `orblk2` in the interface name field.
9. Select **OK** to execute the command.
This saves the symbol as `ORBLK2`.
10. Close the window containing the symbol.
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.
11. In the dialog box, 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. You can then reference the schematics 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.
2. In the dialog box that appears, type `$XILINX_TUTORIAL/calc_sch/andblk2` in the Component Name field.
3. Select **OK**.
A blank schematic sheet appears.

Adding the First Component to a Schematic

1. From the menu bar, select **Libraries** → **XILINX Libraries**.
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 `XC4000E` or `XC9000`.

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.

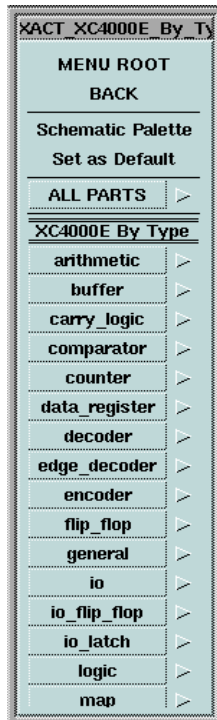


Figure 9-13 XC4000E Library BY TYPE Menu

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. In the small dialog box that 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.

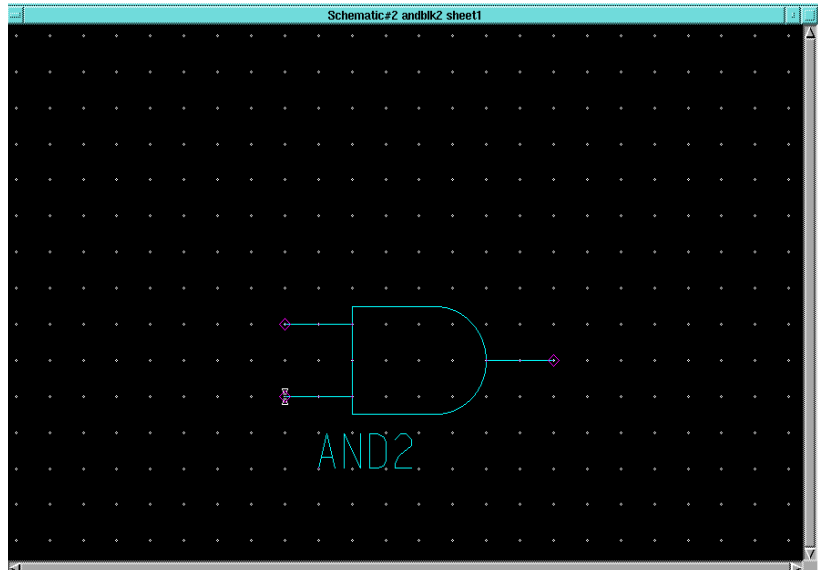


Figure 9-14 Placing a Component

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 appears 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.

1. Using this method, select and place a second `and2` symbol as shown in the following figure.

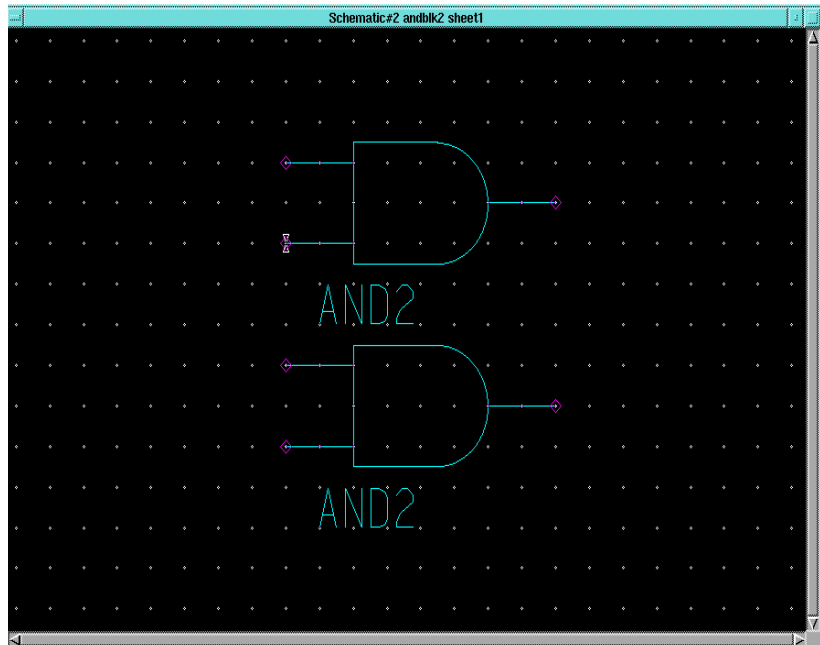


Figure 9-15 Placing a Second Component

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.

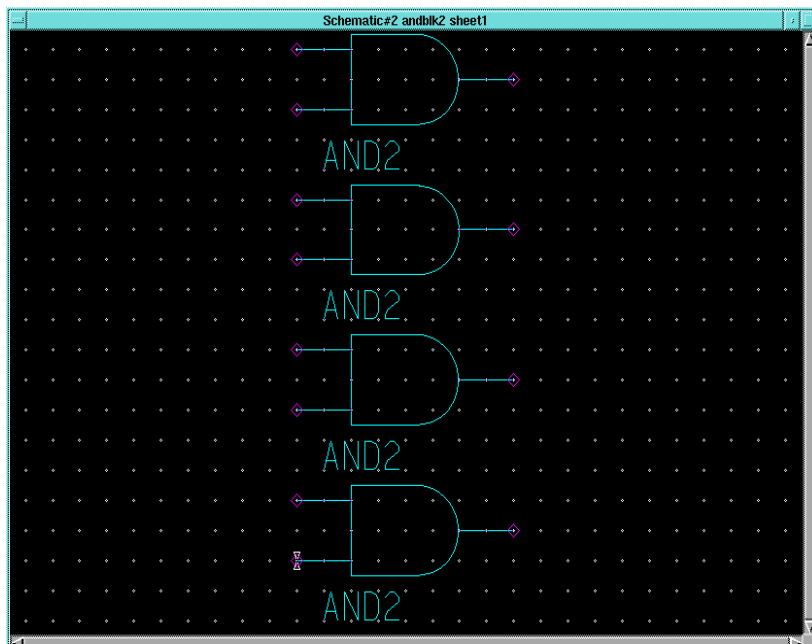


Figure 9-16 Component Placements for ANDBLK2

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.
2. Select the component by clicking on it with the left mouse button.
The component appears highlighted, indicating that it has been selected.
3. Select **Right Mouse Button** → **Move**, or use the stroke 74159.

A small dialog box appears.

4. 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 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.

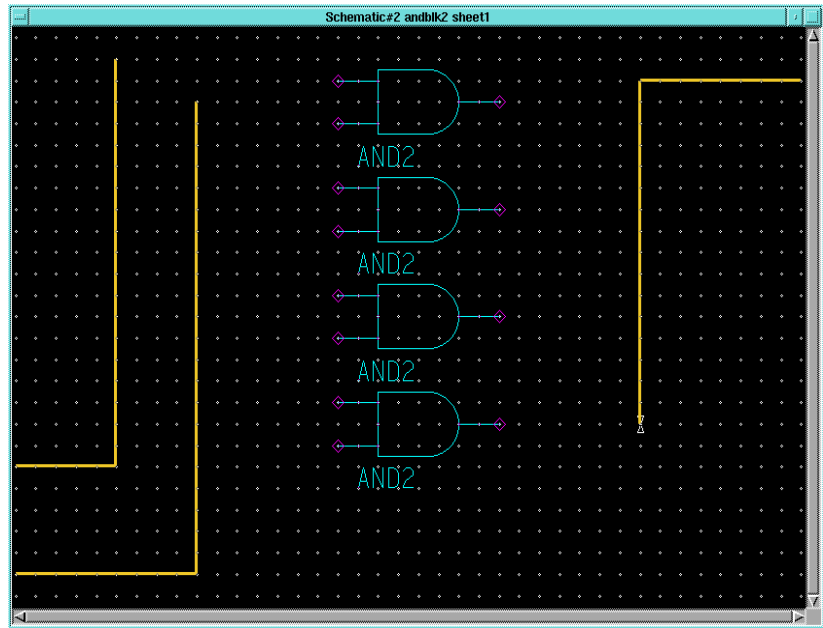


Figure 9-17 ANDBLK2 Schematic with Buses

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.
2. 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, resulting in a different menu appearing on the screen. If this happens, press the F2 key and repeat step one.

3. Move the cursor to the top input pin of the top and2 gate, then click the left mouse button.

4. Move the cursor to the left, so that the pointer is laying atop the leftmost bus. (The wire should form a ninety-degree angle with the bus.) Click the left mouse button twice to terminate the wire.

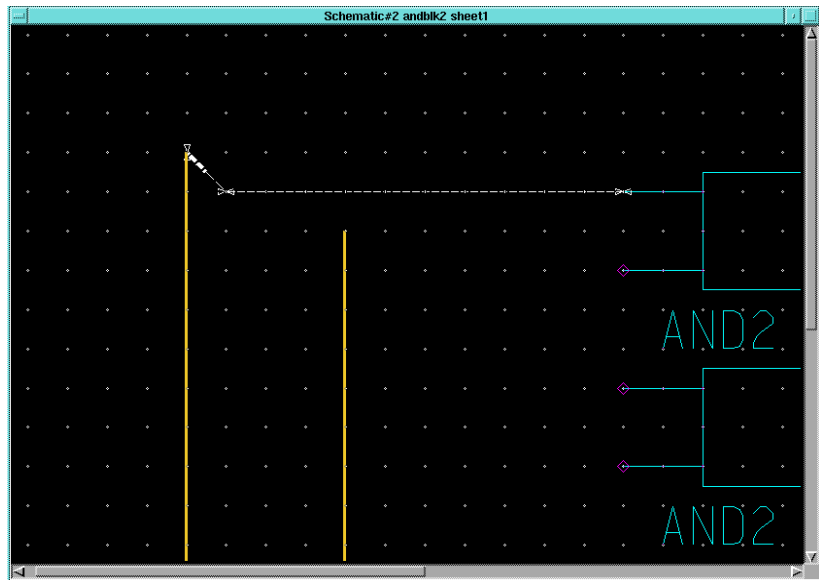


Figure 9-18 Connecting a Net

A bus ripper is inserted automatically between the wire and the bus as shown in the **“Connecting a Net”** figure. 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.

A dialog box as shown in the **“Label Bus Dialog Box”** figure below appears. This box allows you to label a net which has a bus ripper connected to it. 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).

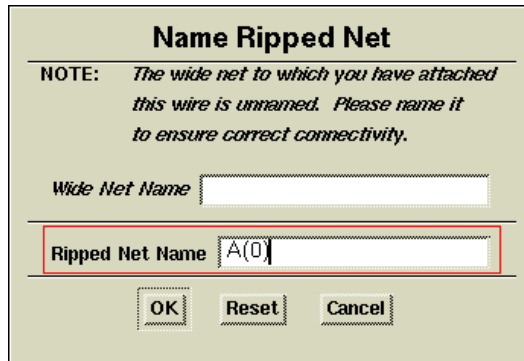


Figure 9-19 Label Bus Dialog Box

Fill out the fields as shown in this example, so that this first net is labeled A(0). Note that you can label the bus as well as the net, but you will do this later.

If the dialog did not appear as shown, choose SETUP RIPPER from the palette and make sure that Ripper Mode is set to "Implicit" and that Ripper Query is set to "On". After setting these parameters, select OK, then delete the net and start again from step one.

5. After filling out the dialog box, you are asked to place the net label, or net name, on the schematic. Place the label as shown in the following figure and click the left mouse button.

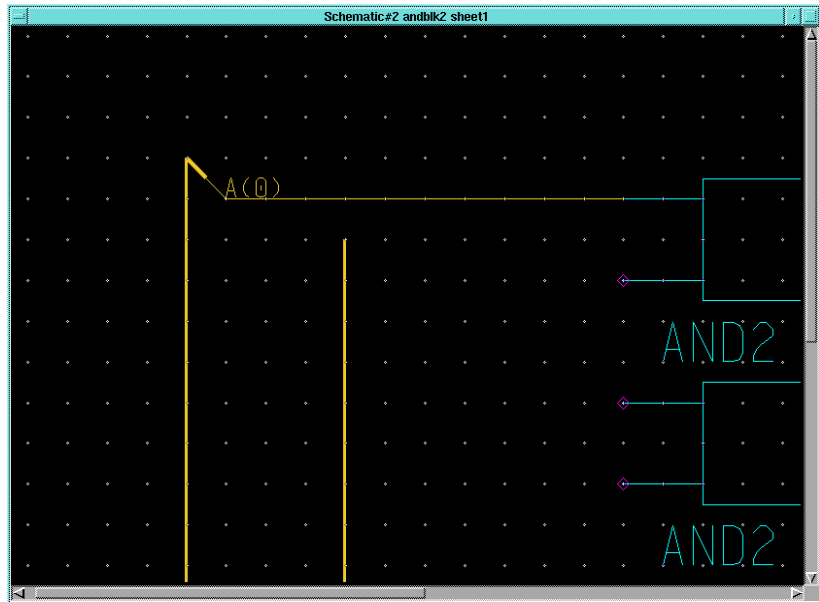


Figure 9-20 Adding and Placing a Net Name

6. Press the Escape key to exit the wire-adding mode.

Completing the Net Connections

Add the remaining nets to the schematic as follows:

1. Press the F3 key to execute the Add Wire command, or use the downward stroke 258.
2. Add the remaining nets as shown in the **“ANDBLK2 with All Wires and Buses Connected”** 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.

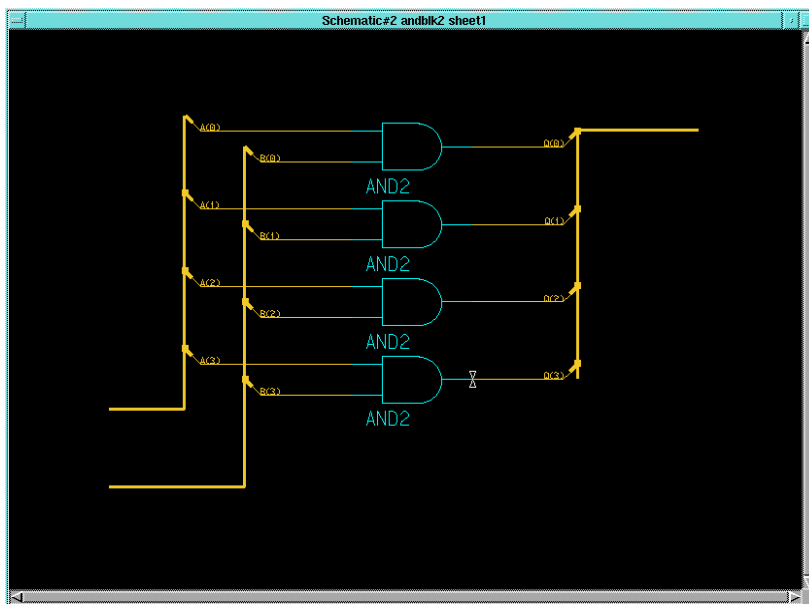


Figure 9-21 ANDBLK2 with All Wires and Buses Connected

Increasing Text Size

At this point, all nets in the ANDBLK2 schematic have been labeled. However, the text of these labels is quite small compared to the other elements in the schematic.

To make the labels more readable:

1. Select all nets in the design by dragging with the left mouse button over the A(x) and B(x) nets then releasing, then doing the same with the Q(x) nets. In each case, a selection marquis appears as shown in the “**Selecting Nets**” figure. This is an additive selection; elements selected previously remain selected. Note that the marquis need only touch elements you wish to select; it need not enclose these elements completely.

Note: If you accidentally select any elements besides the nets (bus rippers, buses, or gates), press F2 to unselect everything, then repeat the selection procedure.

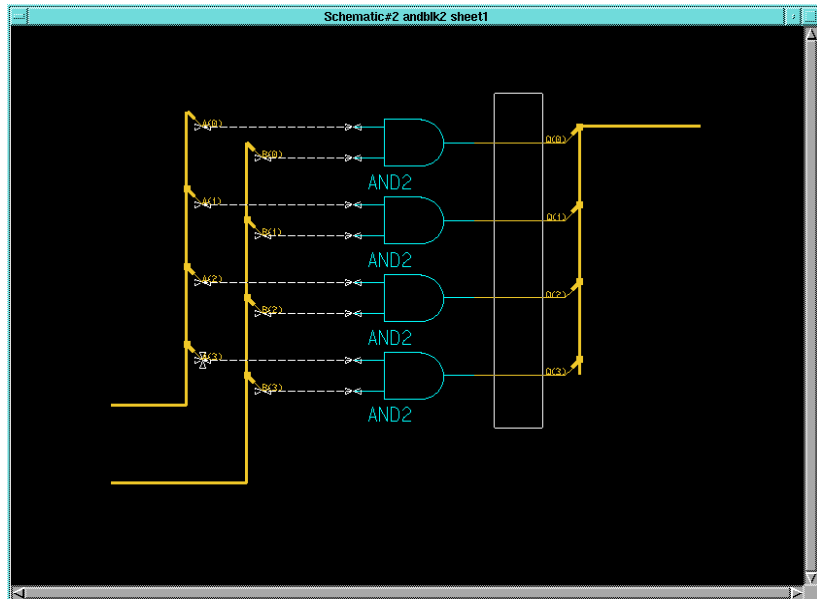


Figure 9-22 Selecting Nets

2. To change the size of the net labels, choose **Right Mouse Button** → **Properties** → **Change Text Height** → **1.0 x Pin Spacing**.
3. In the Change Property Height dialog box that appears as shown below, type **NET** in the Property Name field as indicated to increase the height of all NET properties on all selected elements.



Figure 9-23 Increasing Text Size

All net and bus labels are attached as a property called “NET” with a value equivalent to the net or bus label.

The label sizes are increased as shown in the figure below.

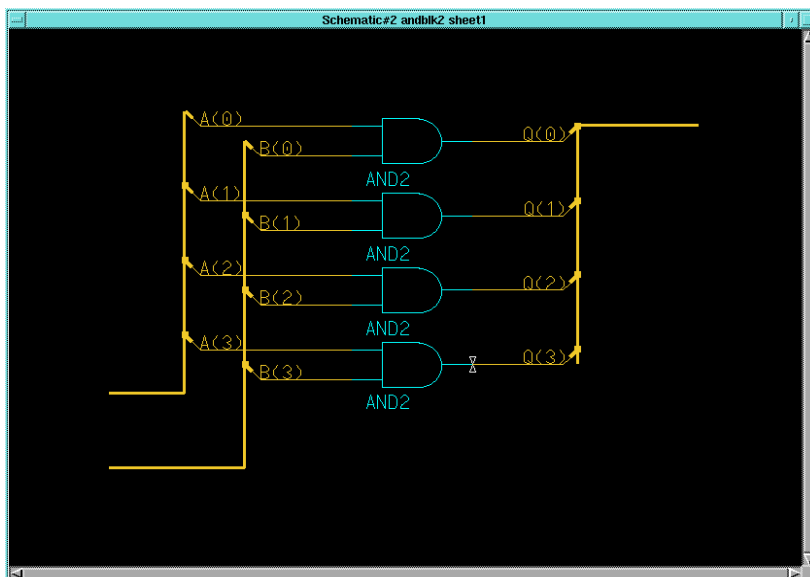


Figure 9-24 Schematic with Larger Net Names

Adding Ports

You must add port symbols 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** → **XILINX Libraries** to select it. 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. 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.

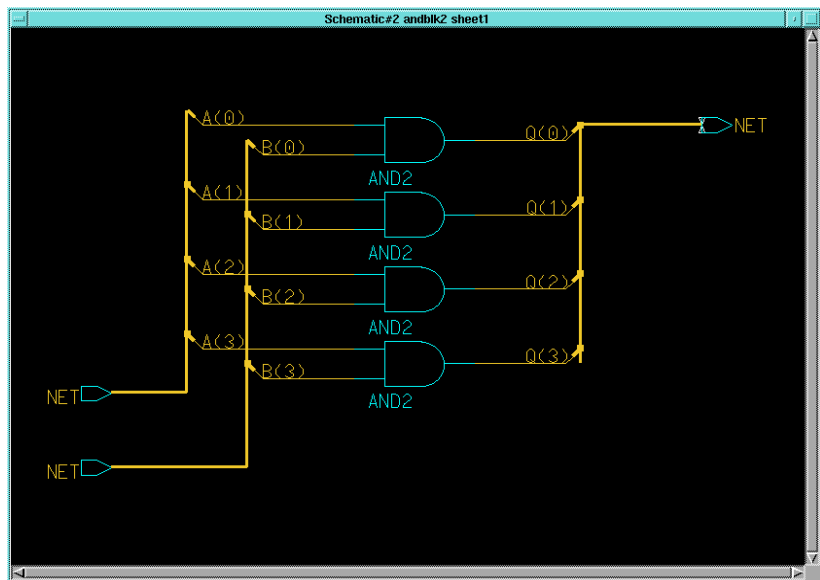


Figure 9-25 Adding Ports

Labeling Ports

Normally, you label nets and buses by selecting them, then executing the menu selection **Right Mouse Button** → **Name Nets** as shown later in this tutorial. However, 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 the **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.

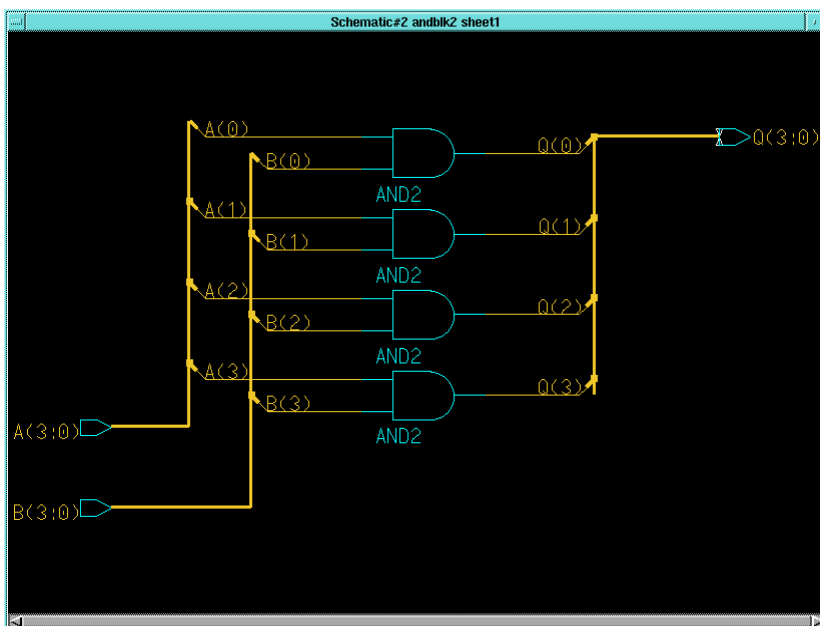


Figure 9-26 Labeling Buses

Saving the Schematic

The schematic is now complete. Check and save the schematic as follows:

1. Select **Check** → **Sheet**.

The text window that appears should not contain any warnings about unnamed or dangling net vertices, since all pins in the schematic should be connected.

2. If these or any other warnings or errors occur, recheck the schematic against the following figure.

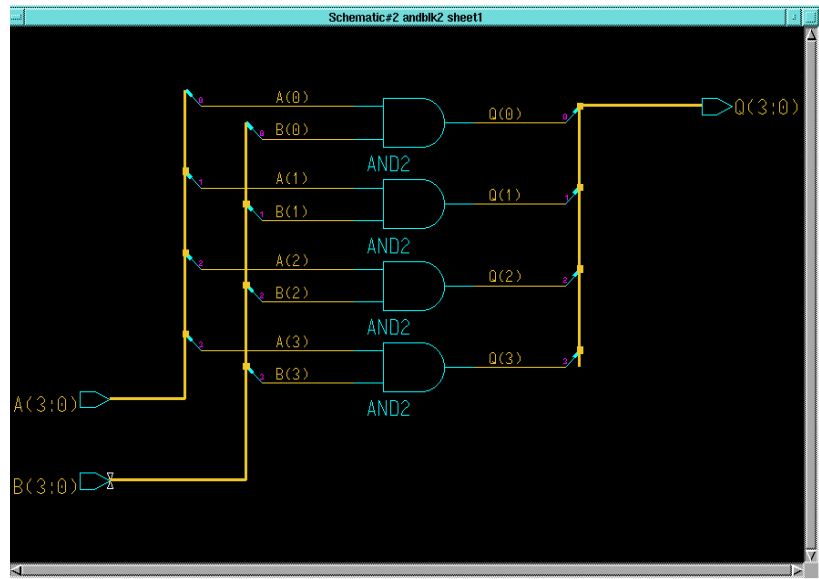


Figure 9-27 Completed ANDBLK2 Schematic

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.

3. Once all schematic errors have been corrected, check the design again if necessary, and close the check sheet text window.
4. 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 as described in the following procedure.

1. Press the **F2** key to unselect everything on the ANDBLK2 schematic.
2. Display the **BY TYPE** library menu and 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.

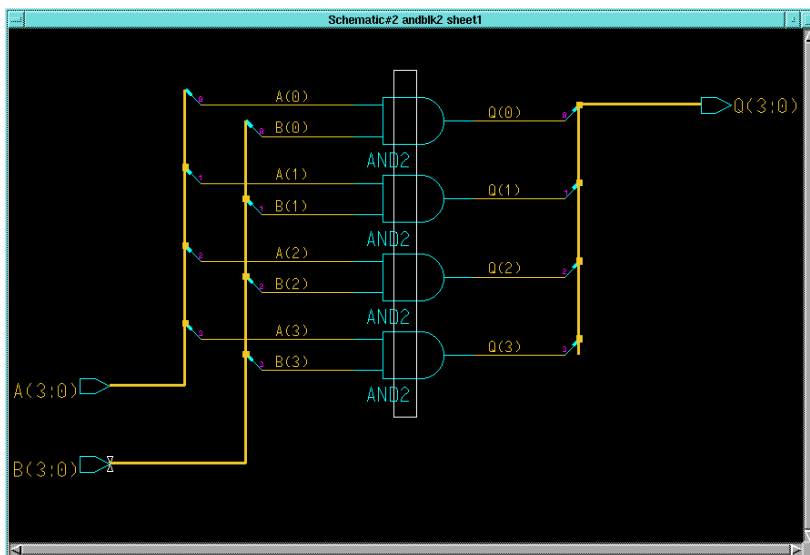


Figure 9-28 Selecting Gates

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**.
10. In the dialog box that appears, type `$XILINX_TUTORIAL/calc_sch/orblk2` in the Component name field and leave all other fields blank.
11. Press return to save the schematic.

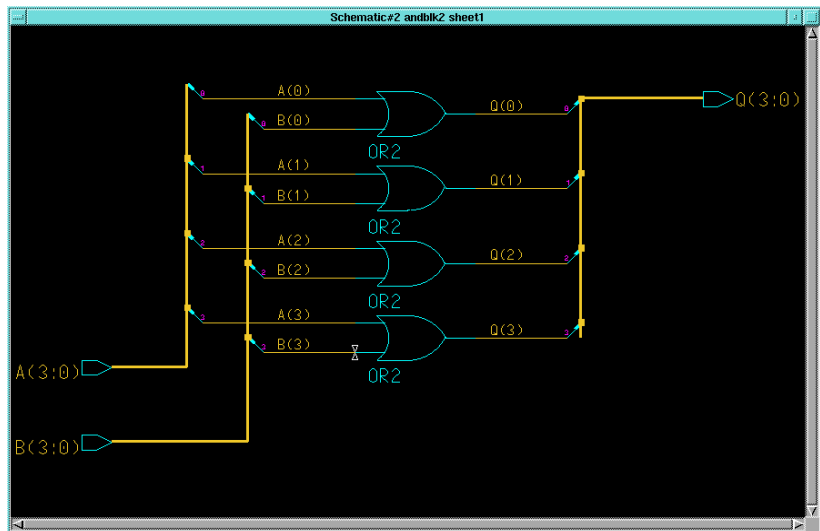


Figure 9-29 Completed ORBLK2 Schematic

Editing 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.

2. In the dialog box that 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.
3. Choose **OPEN SHEET** from the Session Palette.
4. Press the **Navigator** button to open a Navigator window.
5. If necessary, change directories to the \$XILINX_TUTORIAL/calc_sch 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.
6. 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_sch/calc”.
7. 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.
8. Press **F2** key to unselect everything on the schematic.
9. Select the **ALU** symbol.
10. 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 Open Down dialog box appears as shown in the following figure.

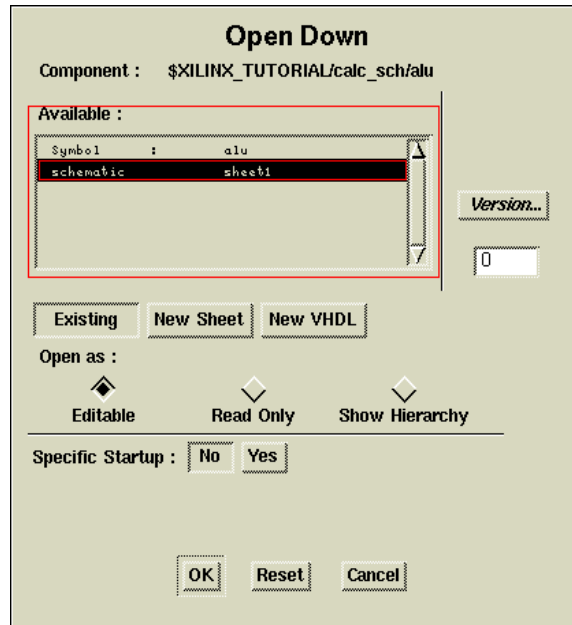


Figure 9-30 Open Down Dialog Box

11. In the Open Down dialog box, specify whether to modify the symbol or the schematic for ALU by selecting **schematic sheet1**, with the left mouse button. The selected line appears highlighted in the dialog box.
12. Press **return** or select **OK**.

A second schematic window appears containing the ALU schematic.

Placing User-Created Components

You can now place the ANDBLK2 and ORBLK2 symbols on the schematic as shown in the figure below. You place the symbols using the same procedure you 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**.

In the Add Instance dialog box that appears, use the Navigator button to select the \$XILINX_TUTORIAL/calc_sch/andblk2 component, or type the name in the Component Name field.

4. Press **return** or select **OK** to execute the command.
5. Move the cursor to the correct location as shown in the following figure.

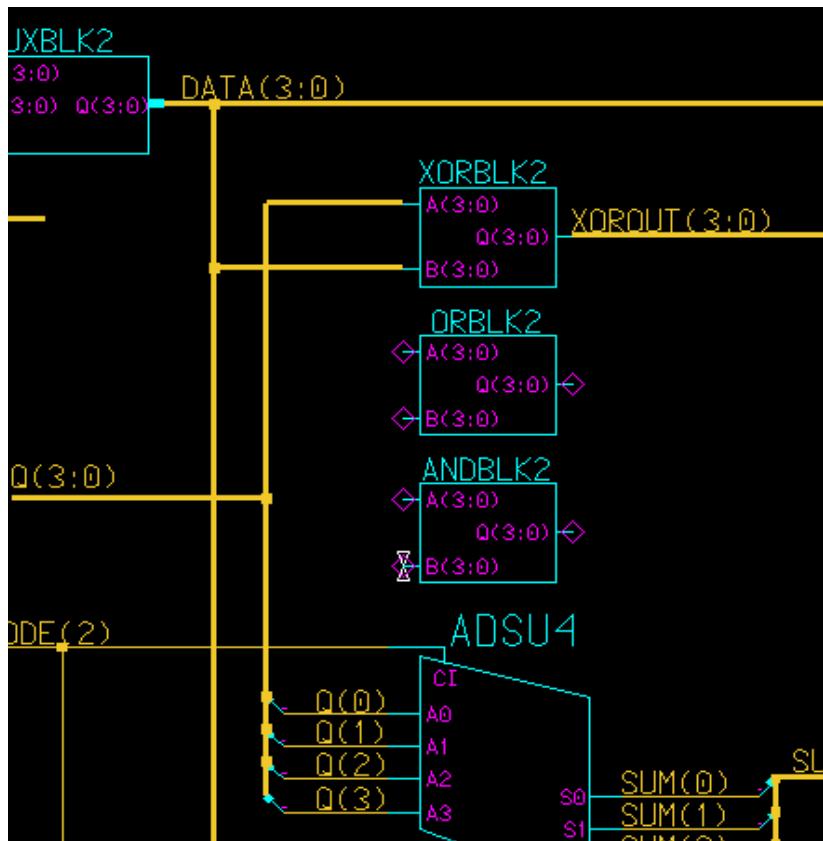


Figure 9-31 Adding ANDBLK2 and ORBLK2 to ALU Schematic

6. Press the left mouse button to place the component.
7. Follow the same procedure to add the ORBLK2 symbol. Refer to the ALU schematic in the figure above for proper placement.

8. If you make a mistake when placing a component, select it (after pressing F2 key) and use **Right Mouse Button** → **Move** to reposition it.

Placing Library Components

The next step in the tutorial is to add the fd4re and and5b2 components to the ALU schematic. The fd4re component is available in the Xilinx Unified Libraries and consists of four flip-flops with a clock enable. The and5b2 component is a five-input AND gate with two inputs inverted (“bubbled,” hence the “b”).

Note: These components are available in all libraries, including those for the XC4000E and XC9000.

1. Use the **shift -F8** keys to display the entire ALU schematic.
1. Use the **F8** key to zoom into the open area in the lower right-hand corner.
2. Select **Libraries** → **XILINX Libraries** 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** → **fd4re** from the Library menu.
5. Move the cursor into the schematic window.
An outline of the fd4re component appears.
6. Move the component to lower right corner of the schematic, approximately to the location shown the **“Adding FD4RE and AND5B2 to ALU Schematic”** figure.
7. Press the left mouse button to place the component.
8. Repeat steps two through six to place the and5b2 component next to the fd4re as shown in the figure below.

When choosing the component from the library menu, use the selection **BY TYPE** → **logic** → **and5b2**.

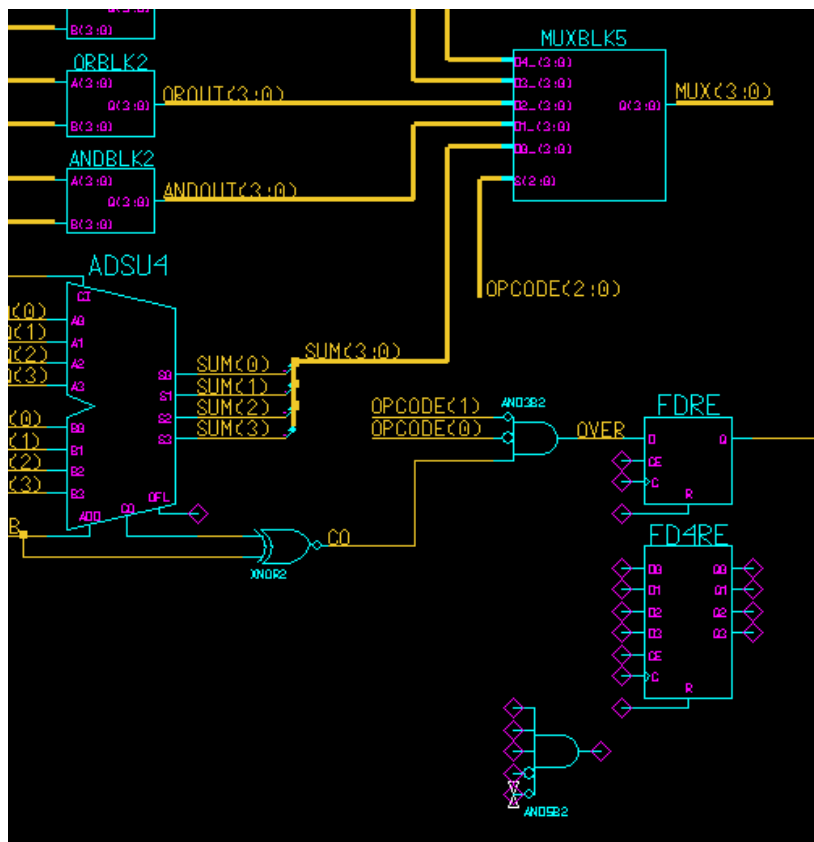


Figure 9-32 Adding FD4RE and AND5B2 to ALU Schematic

Adding Nets, Buses, Ports and Labels

FD4CE and AND5B2

Next, complete adding the fd4re and and5b2 symbols by adding nets, buses, and labels as follows:

1. Add the necessary nets and buses to complete connections for fd4re and and5b2 as you did for the previous schematic.

The figure below displays the labeled nets and buses for fd4re and and5b2.

2. Add ports to the nets and buses attached to the fd4re and and5b2, 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. To add net labels to nets not connected to ports or buses, select the nets as described earlier, then select **Right Mouse Button** → **Name Nets**.

For each selected net, the schematic editor asks you for a net label, then offers you the opportunity to place each label on the schematic. The nets should be labeled as shown.

5. Increase the size of the label text as described earlier.

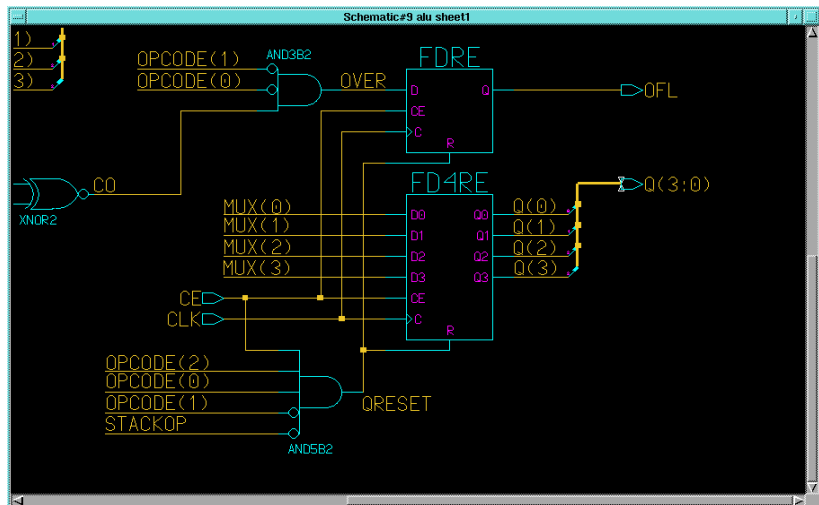


Figure 9-33 Nets, Buses, and Ports for FD4RE and AND5B2

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 by using the same **Right Mouse Button** → **Name Nets** from the previous

section. You only need to label the output buses of the two components, since the inputs to these components are connected to pre-labeled buses.

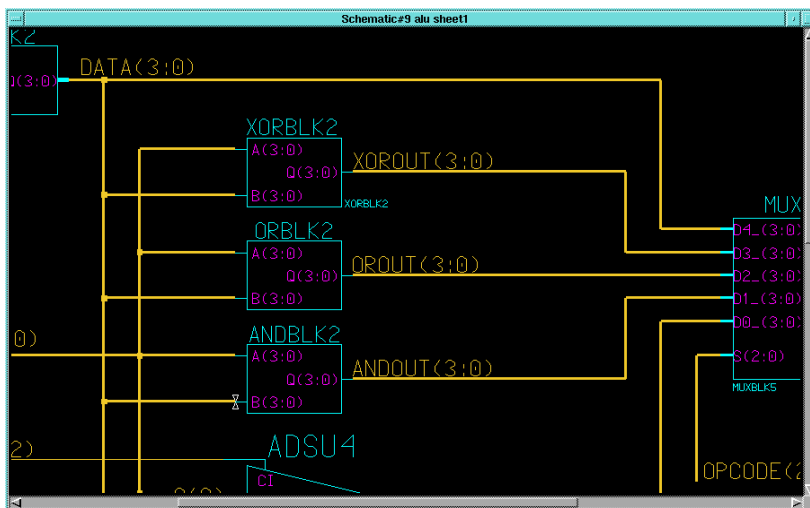


Figure 9-34 Nets, Buses and Labels for ANDBLK2 and ORBLK2

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 ORBLK2 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 **ORBLK2**, then press **return** or choose **OK**.
6. Move the text to position it as shown in the following figure and click the left mouse button to place the text.
7. Label the ANDBLK2 symbol the same way using the label **ANDBLK2**, as shown in the following figure.
8. Give the FD4RE component the label **ALUVAL**.

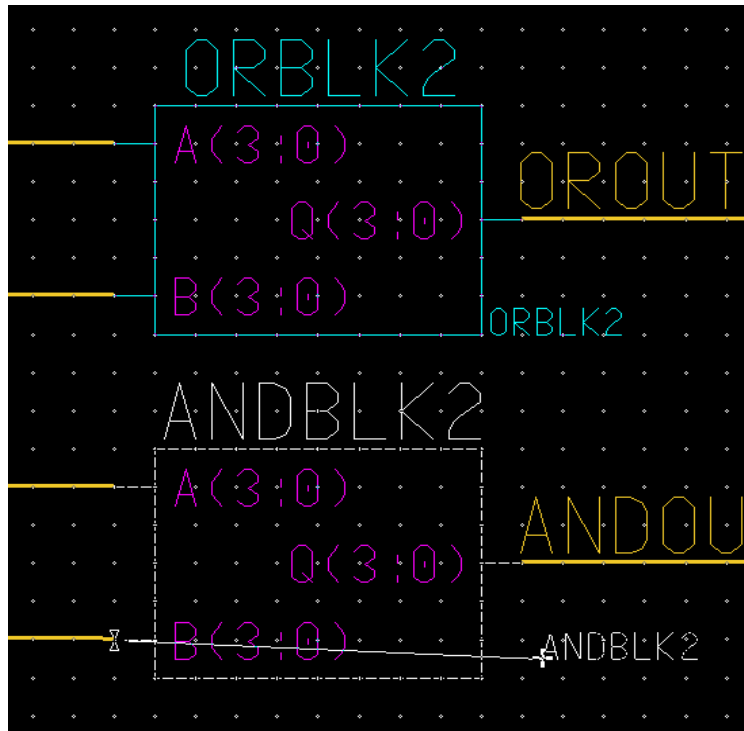


Figure 9-35 Adding Component Labels to ALU Schematic

The completed ALU schematic is shown in the following figure.

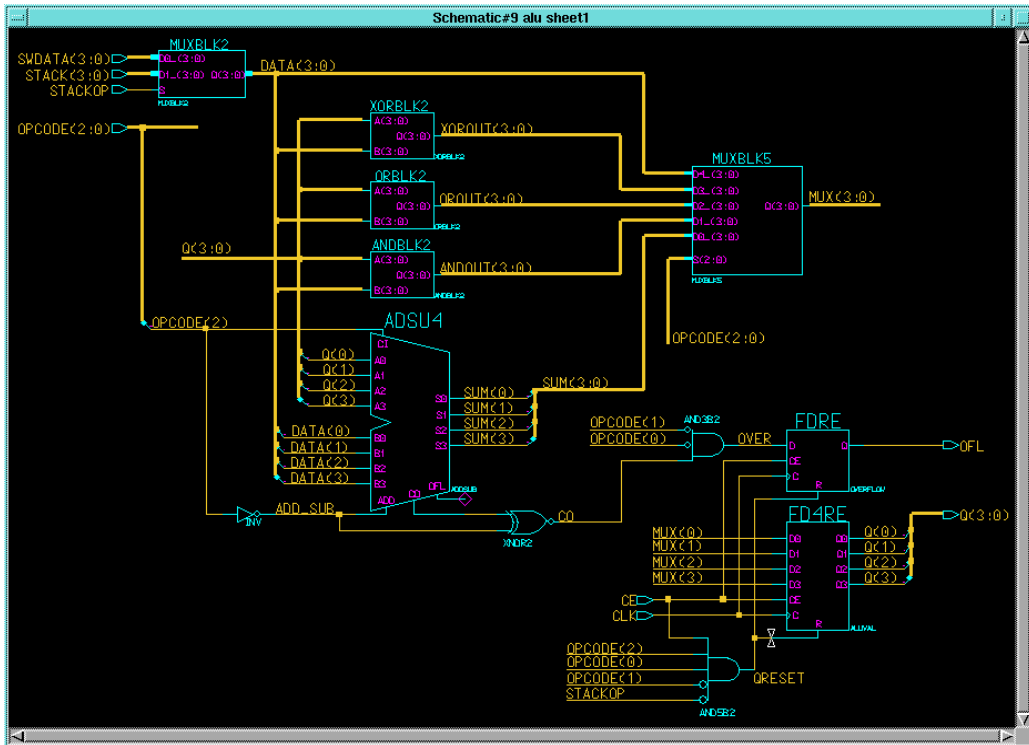


Figure 9-36 Completed ALU Schematic

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 XC4000E 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 fd4re symbol as follows:

1. Press the **F2** key to unselect everything.
2. Select **fd4re** with the left mouse button.
3. Select **File** → **Open** **Down** from the menu bar.
4. In the dialog box that appears, select the schematic sheet and click **OK**.

As shown in the following figure, fd4re consists of four fdre symbols.

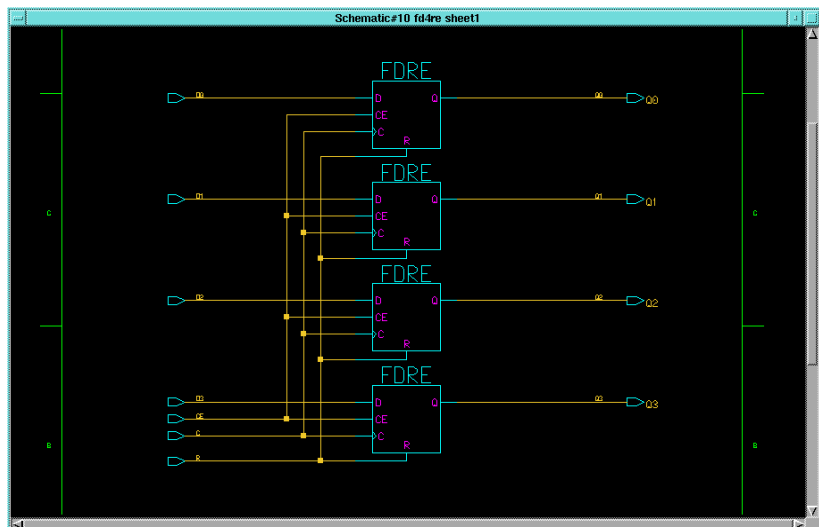


Figure 9-37 FD4RE Schematic from XC4000E Library

Viewing a Xilinx RPM (XC4000-Based Families Only)

Note: The following description of RPMs contains detailed information on the XC4000E architecture. Refer to *The Programmable Logic*

Data Book for more information on the XC4000E CLB structure and fast carry logic.

If your design is not targeted for the XC4000E family, read this section, but do not perform any of the commands. Continue the tutorial with the **“Opening the Calc Schematic” section** (the next section).

Note: The XC5200 library also contains RPMs. If you have an XC5200 schematic, you may open the ADSU4 component as described here to see how this RPM is implemented in that family.

The ALU contains a component from the Xilinx library, adsu4, which is a four-bit wide adder/subtractor. If your design is targeted for the XC4000E library, this schematic is implemented as a Relationally Placed Macro (RPM). If your design is not targeted for the XC4000E or XC4000EX 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 XC4000E 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.

- Use the **F8** key (or stroke 159) to zoom into the upper portion of the schematic as shown in the following figure.

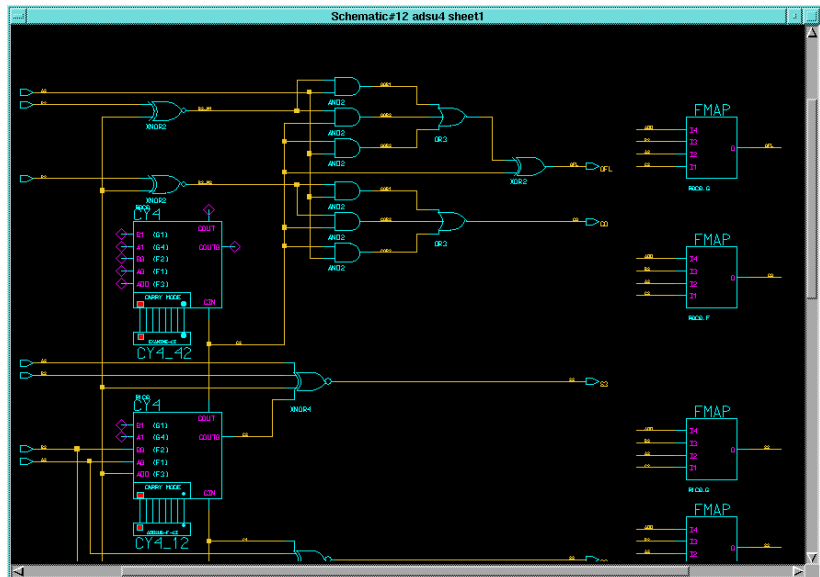


Figure 9-38 Upper Portion of the ADSU4 RPM Schematic

- Press **F2** key to unselect everything.
- Select the FMAP component in the upper right corner.
- Select **Report** → **Object** → **Selected** → **All**.

A text window appears displaying the attributes on the symbol, as shown in the following figure. 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. (You can also call up a report with the 1474123 stroke, which looks like a lowercase “r”.)

Reporting: Instance, Net, Pin, Property, Comment, Frame, Attached pin, Attached property,

Instance	Name	Property	Name	Value	Type
I\$5	\$LCA/xc4000e/fmap/fmap	TS\$703	LIBVER	2.0.0	string
		TS\$700	MODEL	null	string
		TS\$699	COMP	FMAP	string
		TS\$698	INST	I\$5	string
		TS\$697	RLOC	R0C0.0	string
Pins:	Name	Attached	Vertex	of Net	
P\$704	I1		V\$193	N\$21	
	Property	Name	Value	Type	
	TS\$706	PIN	I1	string	
	TS\$705	PINTYPE	in	string	
P\$707	I2		V\$198	N\$22	
	Property	Name	Value	Type	
	TS\$709	PIN	I2	string	
	TS\$708	PINTYPE	in	string	
P\$710	I3		V\$203	N\$23	
	Property	Name	Value	Type	
	TS\$712	PIN	I3	string	
	TS\$711	PINTYPE	in	string	
P\$713	I4		V\$208	N\$24	
	Property	Name	Value	Type	
	TS\$715	PIN	I4	string	
	TS\$714	PINTYPE	in	string	
P\$716	0		V\$212	N\$25	

Figure 9-39 RLOC Attribute on FMAP Component

8. Close the text window to return to the adsu4 schematic window.
9. Use the scroll bars on the sides of the window or double-click the middle mouse button 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. These locations are relative, not absolute.

The macro is not defined as placed in the uppermost CLB in the left most column. Regardless of the RPM's absolute location, the logic associated with the FMAP with the location R0C0 is always at the top, R1C0 is in the CLB directly below, and so on.

10. Close the adsu4 schematic and return to the ALU schematic.

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 XC4000E Oscillator

If your design is not targeted for the XC4000E family, read this section, but do not perform any of the commands.

The XC4000E 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.

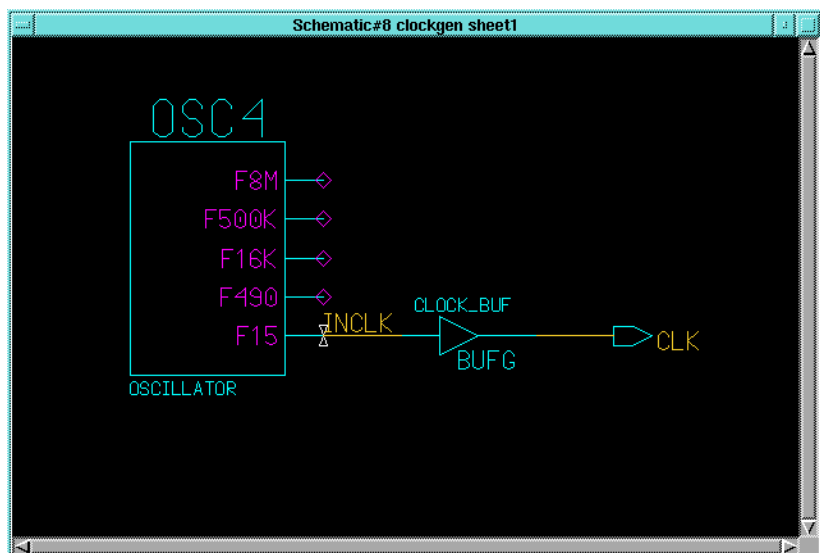


Figure 9-40 CLOCKGEN Schematic

The CLOCKGEN schematic contains an XC4000E 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 XC4000E family designs. The clock output from OSC4 is buffered through a BUFG global clock buffer.

XC4000E 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. A BUFG symbol can represent either type of buffer, and allows the implementation software to choose which type of global buffer is best in each situation. BUFG also facilitates design retargeting to other Xilinx device families, since it can represent any type of global buffer in any family. The BUFG in the Calc design is substituted for a BUFGS during design implementation, because the clock is generated internally by the on-chip oscillator. See the *XACT Libraries Guide* and the *The Xilinx Programmable Logic Data Book* for more information on global clock buffers for Xilinx devices.

Controlling FPGA/CPLD Layout from the Schematic

Assigning Pin Locations

It is highly recommended that you let the automatic placement and routing program, PAR, 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. You should define the initial pinout 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 PAR 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 STACKLED(0) signal on the Calc schematic as follows:

1. Position the mouse over the “PXX” text to the right of the pad attached to net F; this is the default location property attached to the net. Refer to the following figure.

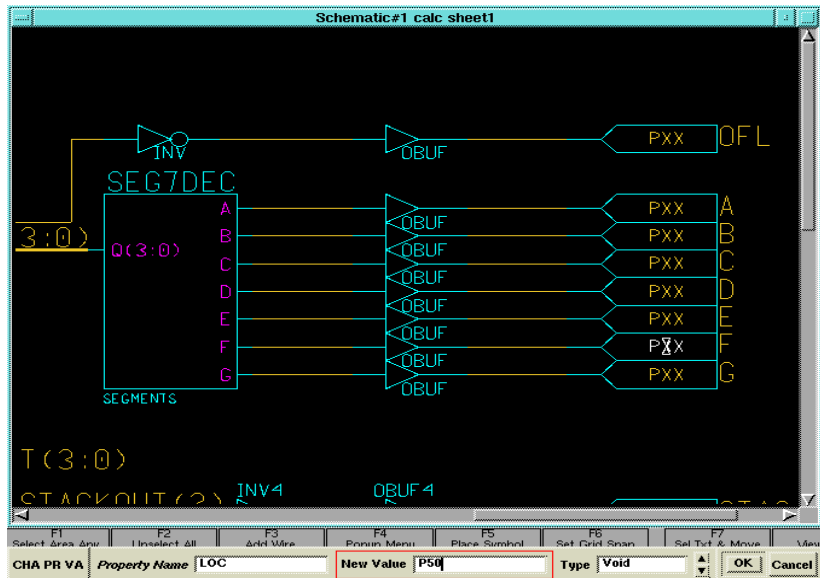


Figure 9-41 Assigning a Location to an Output Net

2. Without moving the mouse, press **Shift-F7**.

3. In the dialog box that appears, modify the “PXX” text to read P50.
4. Click **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, HQFP, and other “numeric-only” package pins are designated with a P followed by the pin number, such as P17. PGA and other grid-array package pins use alphanumeric values such as A12. The Programmable Logic Data Book lists the pinouts of each FPGA and CPLD for each package that Xilinx supplies.

Designating FAST Pads

You can modify the output slew rate by assigning a FAST attribute to the output buffer, as shown in the “**Designating a FAST Pad**” 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. See *The Xilinx 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**.
4. In the dialog box that appears, type the word **FAST** in *both* the Property Name and Property value fields. (This double entry is applied to any property that does not take a value.)
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.

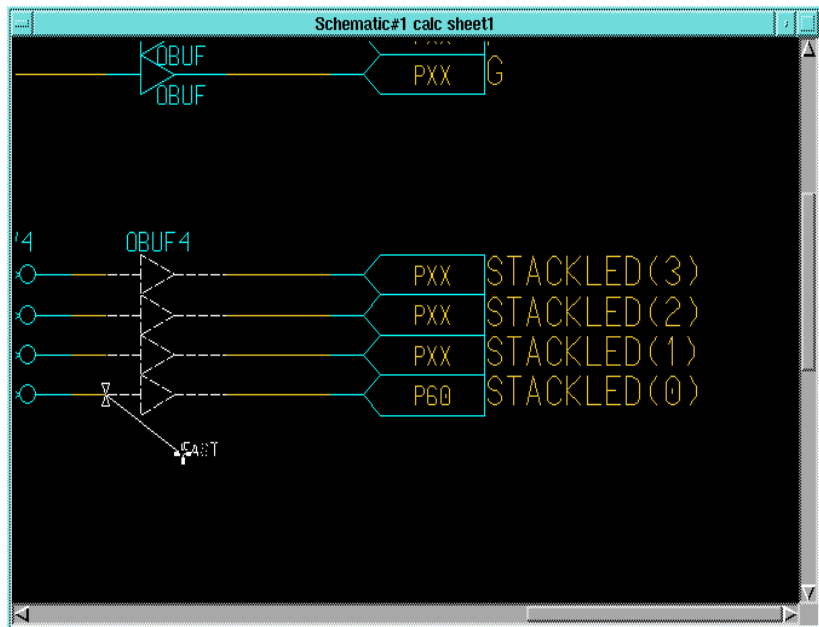


Figure 9-42 Designating a FAST Pad

Using the I/O Flip-Flops

Xilinx XC3000A and XC4000E 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 *Xilinx 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 SWITCH7 macro attached to the input bus SW(7:0) in the lower-left area of the schematic has an underlying schematic that consists of seven IFD (input flip-flop D-type) Xilinx primitives. If similar flip-flops, such as FDs, had been used instead, the flip-flops in the IOBs would be wasted and would occupy valuable CLB resources.

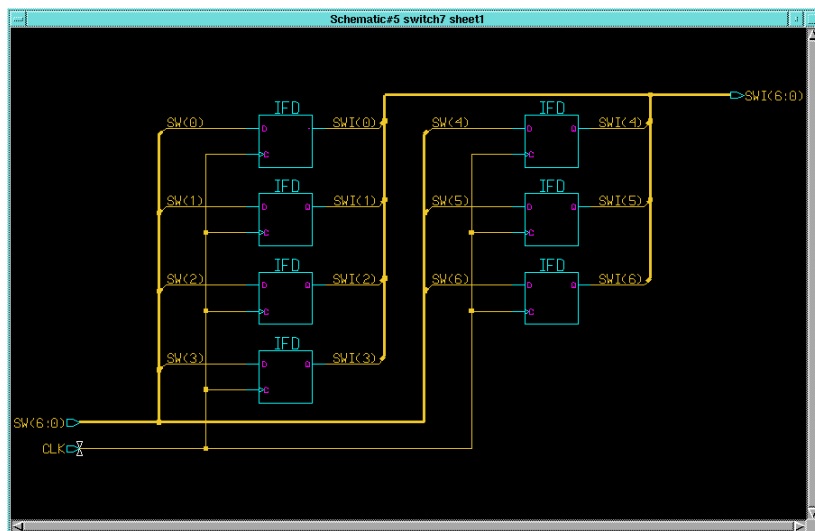


Figure 9-43 SWITCH7 Schematic Using Input Flip-Flops

Saving the Calc Schematic

Before continuing, check and save the changes made to Calc, as shown earlier in this tutorial.

Modifying the Design for Non-XC4000E/EX Devices

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 only in an XC4000E or XC4000EX device. This is because these devices have several advanced features not found in other Xilinx device families. Two of these advanced features are the on-chip memory built into the XC4000E CLB and wide-edge decoders.

RAM Stack Implementation

The RAM stack is implemented using a 16x4 RAM macro from the XC4000E 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. In this case, with a stack only four words deep, using the static memory feature of the XC4000E CLB still reduces the stack from eight CLBs to two CLBs.

To view the XC4000E stack implementation, follow these steps:

1. Make sure the STACK symbol is selected in the Calc schematic, and select **Right Mouse Button** → **Open Down**.
2. Choose the schematic to modify and click on **OK**.
3. On the stack schematic is a RAM16X4S component, which represents four 16x1 synchronous RAMs. Select this component and **Open Down** into its schematic.

The schematic for RAM16X4S is shown below.

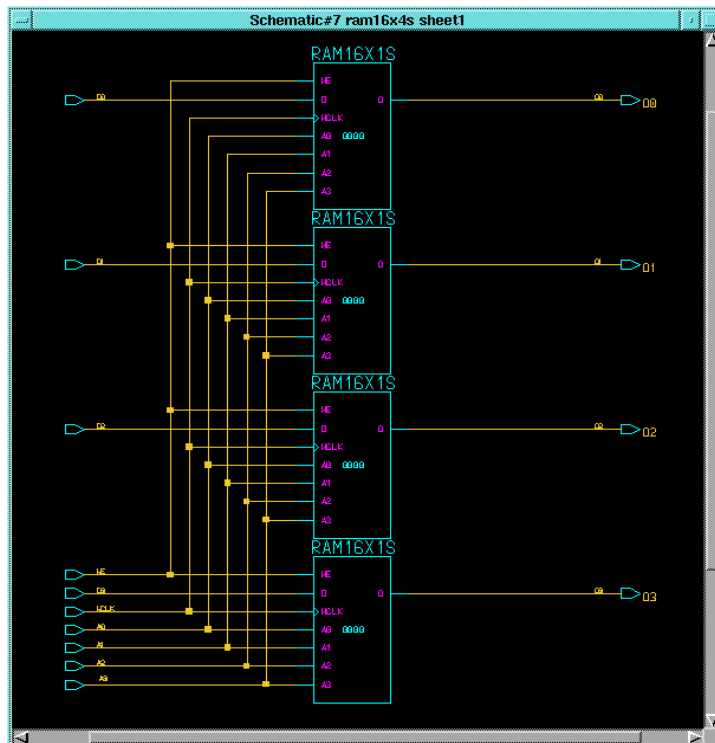


Figure 9-44 RAM16X4S, XC4000E Implementation

Using the Device-Independent Register File

The device-independent stack is implemented by replacing the RAM16X4S with a register file that emulates a synchronous RAM with a set of flip-flops and multiplexers. This implementation can be used for any Xilinx device, even one from the XC4000E family.

If you are targeting an XC4000E device, you may skip this section to take advantage of the RAM feature of the XC4000E.

Make the stack a device-independent schematic as follows:

1. Return to the stack schematic and use the left mouse button to select the RAM16X4S.
2. Select **Right Mouse Button** → **Replace** → **Other**.
3. In the Replace Instance dialog box that appears, fill out the fields as indicated in the following figure and click **OK**.

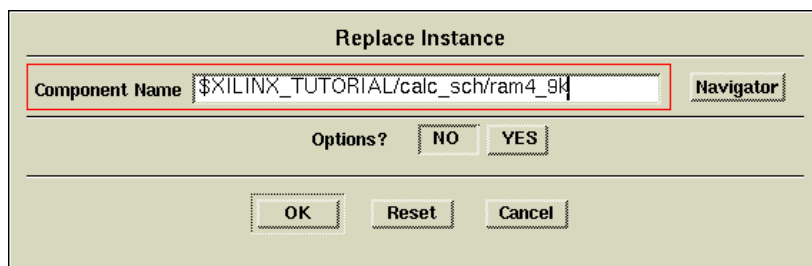


Figure 9-45 Replace Instance Dialog Box

The RAM16X4S is replaced with the device-independent RAM4_9K as shown below. Note that the label RAM_BLOCK has been removed by Design Architect. Add an INST property to this component in the same manner as was done to the components in the ALU schematic.

Note: If you are targeting a device family other than the XC9000, be sure to run Convert Design on the RAM4_9K schematic as described before.

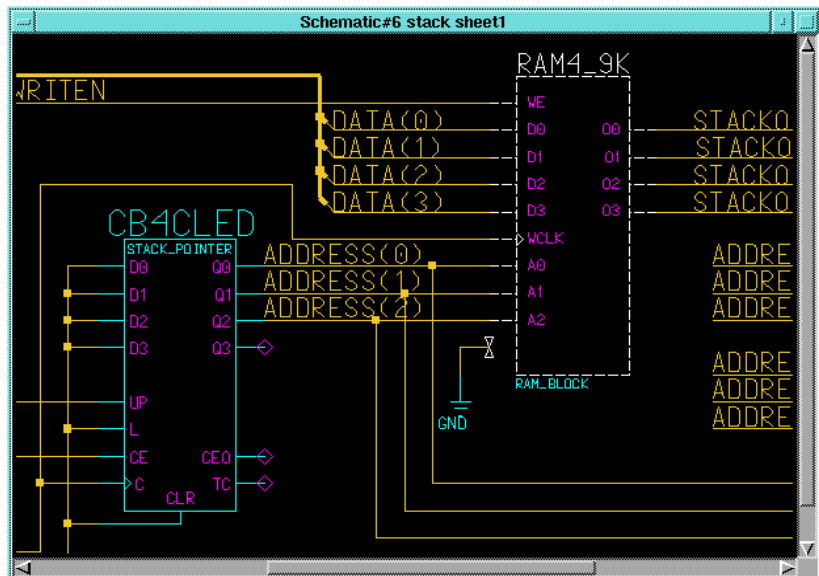


Figure 9-46 Device Independent RAM4_9K

The unused A3 pin that exists on RAM16X4S does not exist on RAM4_9K. Although the detached GND symbol and net are trimmed during the implementation process, you can clean-up the schematic by deleting them.

4. To clean-up the schematic, unselect everything by pressing **F2**, then select **Right Mouse Button** → **Delete** → **Selected** (or use the stroke 741236987, which looks like an uppercase “D”).
5. Check and save the updated stack schematic.

Removing the XC4000E Oscillator

If you are targeting the Calc design to the XC9000 family (or other non-XC4000 family), you must remove the CLOCKGEN circuitry, which includes the OSC4 component, and replace it with an external source.

Note: The XC3000 and XC5200 families also have internal, on-chip oscillators. See the CLKGEN3K and CLKGEN5K components to see how these are used. You may choose to replace the CLOCKGEN component with one of these alternative macros with the Replace

selection from the Instance pop-up menu, instead of following the instructions below.

1. On the Calc schematic, press **F2** to make sure nothing else is selected.
2. Select the CLOCKGEN component with the left mouse button.
3. Delete it by selecting **Right Mouse Button** → **Delete** → **Selected** (or use the stroke 741236987, which looks like an uppercase “D”).
4. Add components, nets, and labels as shown in the following figure. The IPAD symbol may be selected from the library menu under **BY TYPE** → **io**, while the BUFG symbol may be selected under **BY TYPE** → **buffer**.
5. Check and save the Calc schematic.

Since the CLK signal is now sourced by a pad, it must be generated externally.

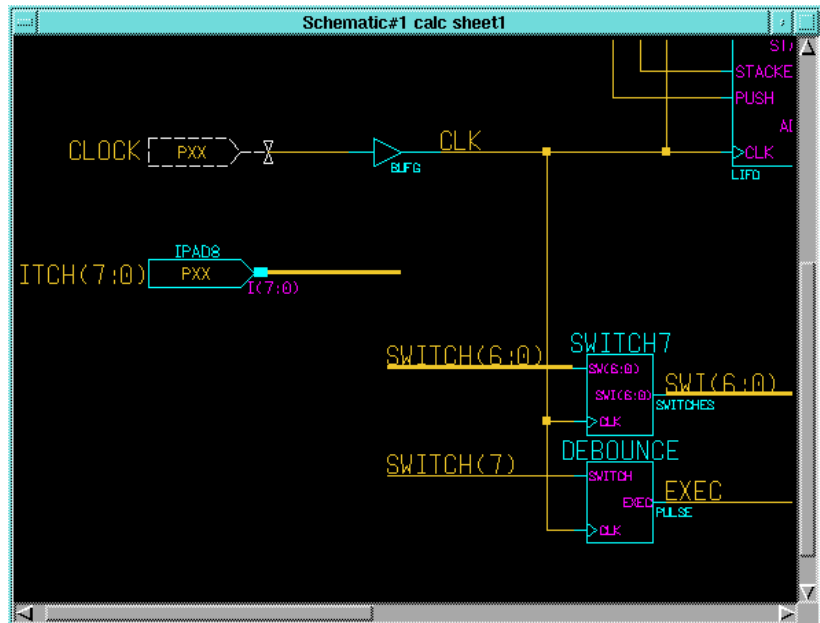


Figure 9-47 Device-Independent Clock Source

Using LogiBLOX

LogiBLOX is a tool that allows you to quickly synthesize modules for common functions such as adders, counters, and multiplexers. It allows you to create components of arbitrary bus width (*e.g.*, a 17-bit adder) and automatically uses the best architectural resources for a particular target device family. In this optional section, you replace the ADSU4 component in the ALU schematic with a LogiBLOX adder. If you choose to leave the ALU schematic in its original form, read this section but do not make save any changes.

Creating and Instantiating a LogiBLOX Module

To replace the ADSU4 symbol with a LogiBLOX module:

1. Bring the ALU schematic into view.
2. Select the **ADSU4** component.
3. Select **Right Mouse Button** → **Delete** to delete the symbol from the schematic.
4. Select **Libraries** → **XILINX Libraries** from the menu bar.
5. From the Xilinx Unified Libraries menu, select **LogiBLOX**.
The Create/Modify/Instantiate LogiBLOX Symbol dialog box appears.
6. In the Symbol component field, type **addsub4**.
This is the user-given component name for the new LogiBLOX-generated module.
7. Under Instantiate symbol, choose **YES**.
This gives you the opportunity to place the component on the schematic immediately after LogiBLOX exits.
8. For PLD Technology, select the family to which you are targeting the design, *e.g.*, XC4000E.

The dialog box should now appear as shown in the following figure.

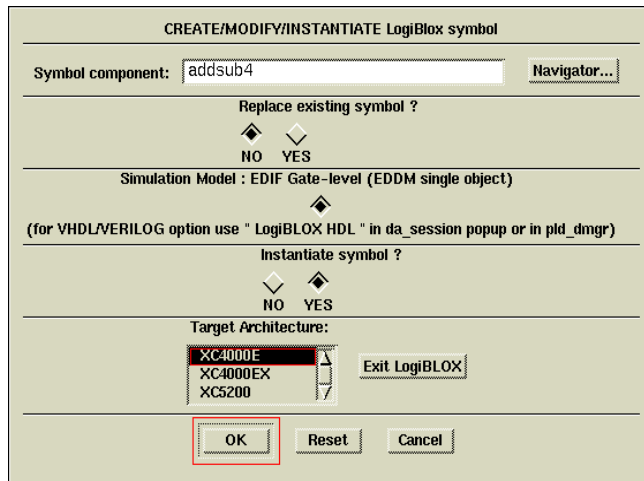


Figure 9-48 Instantiate LogiBLOX Dialog Box

9. Click **OK** to invoke the The LogiBLOX program.

In about 1 minute, the LogiBLOX Module Selector appears.

10. Set the options in this dialog box as shown in the following figure.

You are making a non-registered adder/subtractor module of four bits.

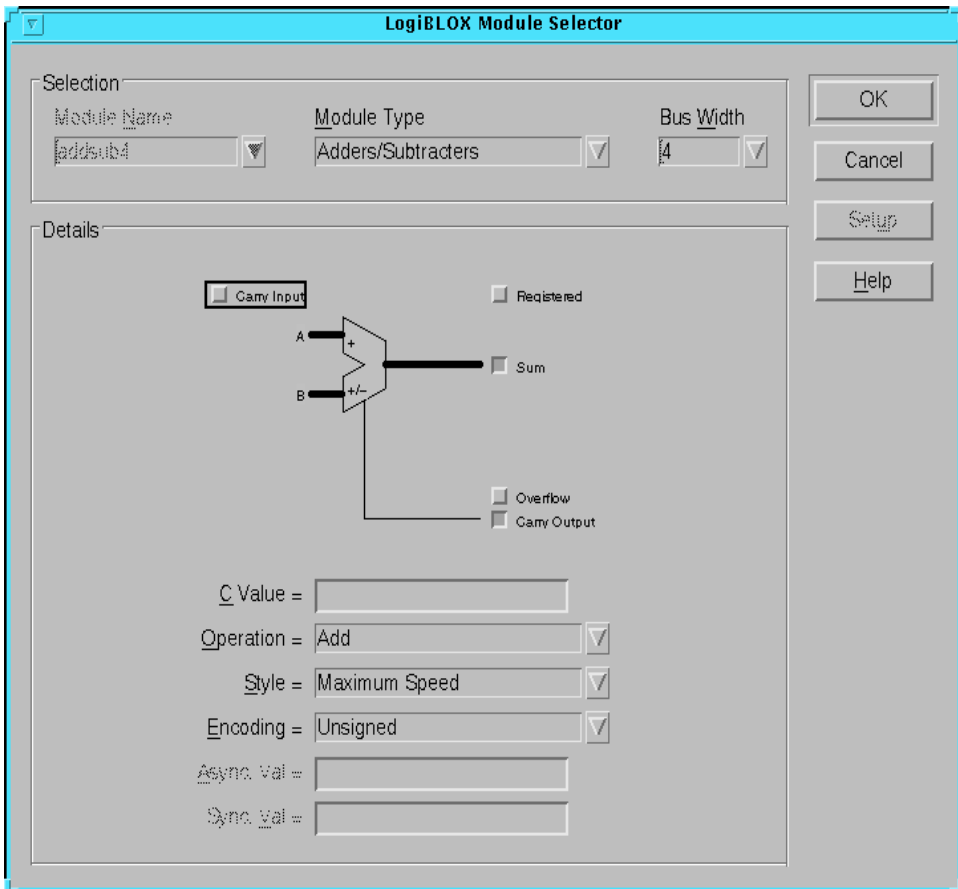


Figure 9-49 Using the LogiBLOX Module Selector

11. Click **OK**.

Design Architect takes a minute or so to generate a symbol for this new module.

12. When the module appears on your screen, place it in the space left by the ADSU4.

Do not worry about lining up pins with nets right now.

13. Arrange the surrounding nets as shown in the figure.
14. Check and save the ALU schematic.

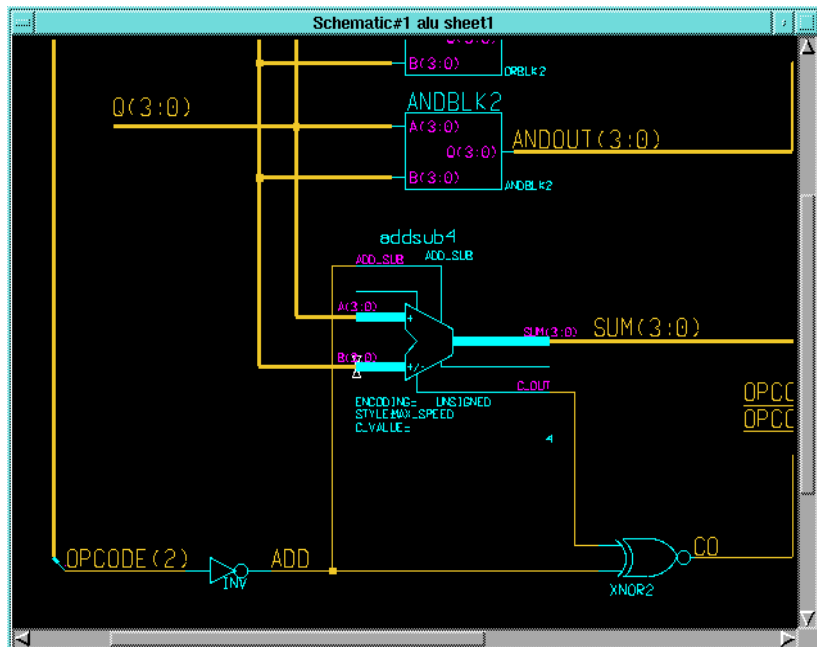


Figure 9-50 Adding the ADDSUB4 LogiBLOX Component

Other Special Components

In this section, you complete the Calc design by adding a **STARTUP** symbol to make the logic resettable and a **CONFIG** symbol to specify the Xilinx part number on the schematic.

The **STARTUP** Block (Optional: XC4000E/EX and XC5200 only)

The **STARTUP** block allows you to globally control different aspects of a design. This example uses **STARTUP** to connect an external signal to the global set/reset net built into the XC4000 family and XC5200 architectures. This global net connects to all flip-flops in the device and sets or resets them asynchronously. (Set or reset is determined at the flip-flop level.) An advantage to using this built-in resource is that no routing resources are wasted tying a system-wide reset signal to all flip-flops in the design. For more information on **STARTUP**, see the *Xilinx Libraries Guide*.

The STARTUP symbol is used here to implement a system-wide reset signal called NOTGBLRESET. This signal is active-low; therefore, when NOTGBLRESET is low, the Calc circuitry is reset.

1. In the Calc schematic, add the components, nets, and labels as shown in the following figure.

You may take IPAD and IBUF from the **BY TYPE** → **io** section of the Xilinx Library, INV from **BY TYPE** → **logic**, and STARTUP from **BY TYPE** → **general**.

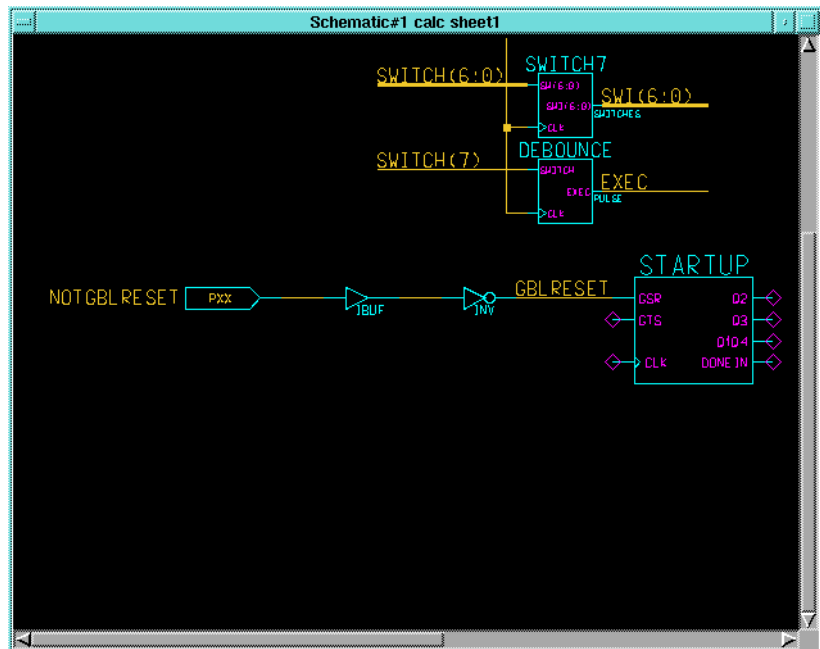


Figure 9-51 Adding the STARTUP Symbol

An inverter is added to the signal path since the GSR pin on STARTUP is active-high. Also, since GSR is *implicitly* connected to all reset logic throughout the device, GBLRESET is connected only to the GSR pin on the schematic.

Note: If you target an XC5200 device, connect your chip-wide reset signal to the GR pin on the STARTUP module.

2. Check and save the Calc design.

Adding the CONFIG Symbol (Optional)

The CONFIG symbol tells the place-and-route software how to process the design. This example uses CONFIG to specify the part number for this device.

To add the CONFIG symbol, follow these steps.

1. From the Calc schematic use the **BY TYPE** → **general** menu to call up the CONFIG symbol from the Xilinx Library.
2. Place this symbol in the lower-right hand corner of the Calc schematic.
3. With the CONFIG symbol still highlighted, select **Right Mouse Button** → **Properties** → **Add** → **Add Single Property**.
4. In the Add Property dialog box, enter the Property Name as “PART” and the Property Value as “XC4003E-4-PC84” and click OK.

This specifies an XC4003E device with -4 speed grade (approximately 4 nanoseconds delay through a CLB) in an 84-pin PLCC.

The PART value may take one of the following two formats:

[XC] *part_number-speed-package*
[XC] *part_number-package-speed*

Therefore, the following values for the PART property are all legal:

XC4003E-4-PC84 (recommended)
XC4003E-PC84-4
4003E-4-PC84
4003E-PC84-4

Note: If using a different device, type that device number into the Property Value field instead, *e.g.*, XC95108-10-PC84

5. Place the property text within the CONFIG symbol as shown in the following figure.

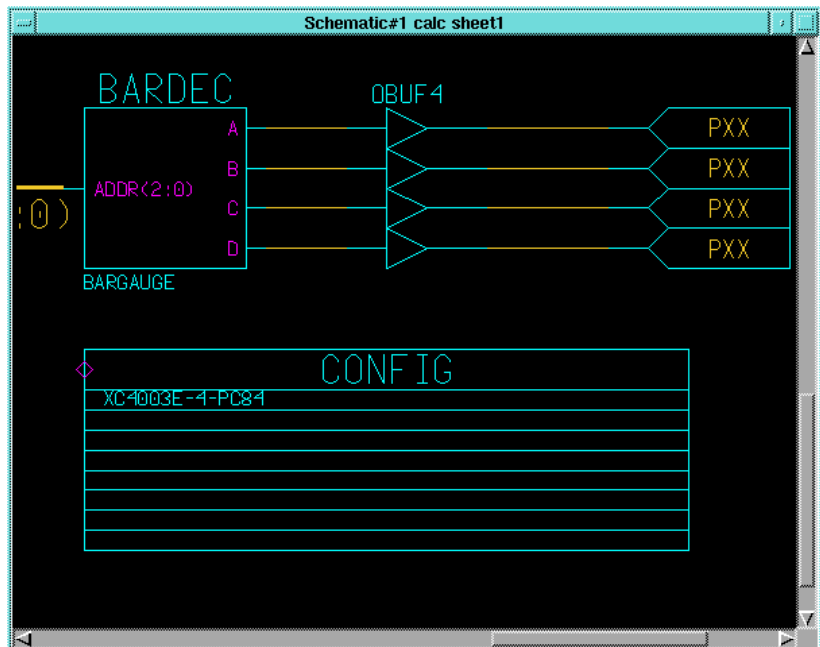


Figure 9-52 Adding the CONFIG Symbol

6. Check and save the Calc schematic.

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.

It is necessary instruct the place and route software to read and apply the .ucf file when the Xilinx Design Manager reads the design. The procedure for doing this is detailed later in the [“Using the Xilinx Design Manager”](#) section.

The calc_4ke.ucf user constraints file which is supplied with this tutorial is shown below as an example of a constraints file. The constraints file syntax is the same for all device families. Since you only specified one pin location for one of the many inputs and outputs on the Calc schematic, you must use a constraints file to place the rest.

```
# CALC_4KE.UCF
# User constraints file for CALC, XC4003E-PC84
# If the F pin is not constrained on the schematic,
# remove the comment (#) from NET F LOC=P50;

NET SWITCH(7)      LOC=P19;
NET SWITCH(6)      LOC=P20;
NET SWITCH(5)      LOC=P23;
NET SWITCH(4)      LOC=P24;
NET SWITCH(3)      LOC=P25;
NET SWITCH(2)      LOC=P26;
NET SWITCH(1)      LOC=P27;
NET SWITCH(0)      LOC=P28;

NET A              LOC=P49;
NET B              LOC=P48;
NET C              LOC=P47;
NET D              LOC=P46;
NET E              LOC=P45;
# NET F           LOC=P50;
NET G              LOC=P51;
NET OFL           LOC=P41;

NET GAUGE(3)       LOC=P61;
NET GAUGE(2)       LOC=P62;
NET GAUGE(1)       LOC=P65;
NET GAUGE(0)       LOC=P66;

NET STACKLED(3)    LOC=P57;
NET STACKLED(2)    LOC=P58;
NET STACKLED(1)    LOC=P59;
NET STACKLED(0)    LOC=P60;

# Remove the NOTGBLRESET line if STARTUP
# is not used in the schematic

NET NOTGBLRESET    LOC=P56;
```

Performing Functional Simulation

You perform functional simulation before design implementation to verify that the schematic that you have designed is logically correct. All components in the Calc design, even the non-schematic Logi-BLOX module, have built-in simulation models so little pre-processing is necessary. However, every top-level design in Mentor

Graphics must have a simulation viewpoint before you can use it in QuickSim. The viewpoint describes how a design should be interpreted, including what components in the design are primitives, as well as how components within the design hierarchy should be modeled.

Using Pld_dve

You use the PLD Design Viewpoint Editor to generate a design viewpoint to tell QuickSim how to interpret certain Xilinx-specific design properties. Follow these steps to generate a viewpoint with pld_dve.

1. Select the calc design object from the appropriate directory in the Navigator window.
2. Invoke pld_dve on the design by selecting **Right Mouse Button** → **Open** → **pld_dve**.

A dialog box appears. Note that the component name, Calc, is entered automatically with a fully qualified path.

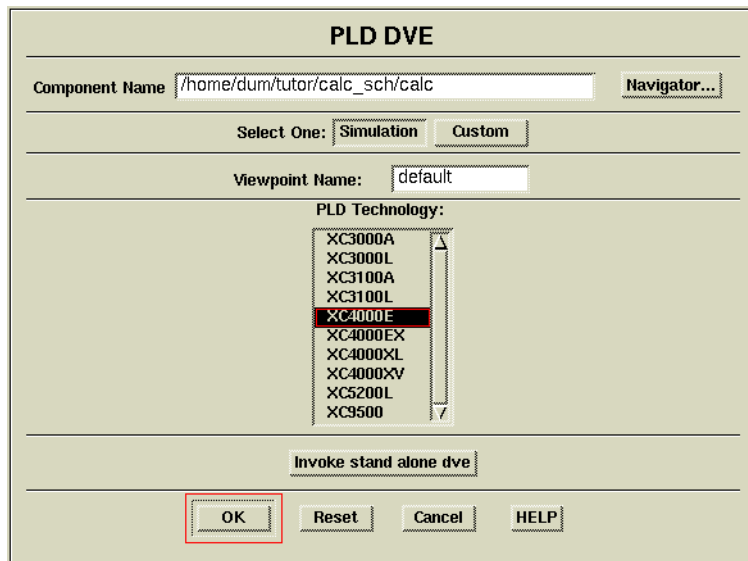


Figure 9-53 Invoking Pld_dve for Functional Simulation

3. Select the appropriate PLD Technology from the listing, e.g., XC4000E, as shown in the figure above. (Leave other options set to their defaults, as shown in the figure.)

4. Click **OK** to execute the `pld_dve` script.
5. Once `pld_dve` completes, dismiss the shell window in which it executed.

Invoking Pld_quicksim

Invoke `pld_quicksim` for functional simulation on the Calc design using the following method:

1. Select the Calc design object in the Navigator window.
2. Invoke `pld_quicksim` on the design by selecting **Right Mouse Button** → **Open** → `pld_quicksim`.

A dialog box appears. Note that the component name, Calc, is entered automatically with a fully qualified path.

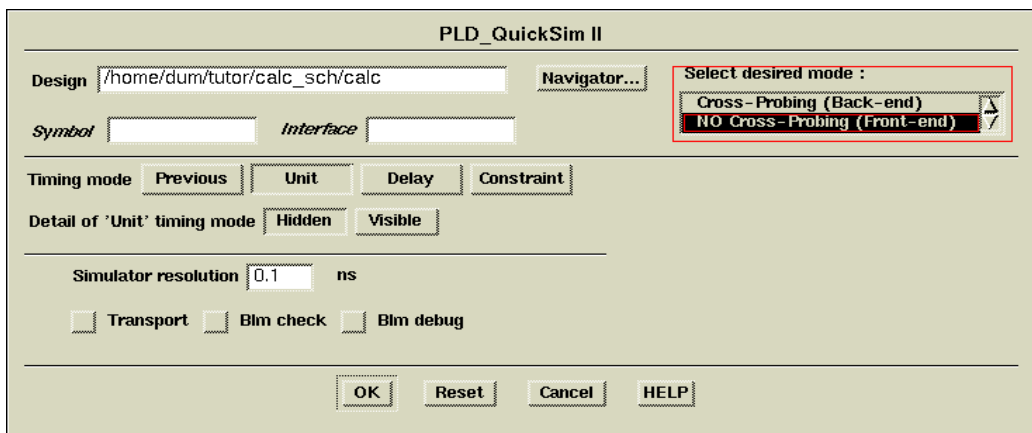


Figure 9-54 Invoking Pld_quicksim for Functional Simulation

3. Under Select desired mode, select **NO Cross-Probing (Front-end)**.

This runs `pld_quicksim` in functional simulation mode, which uses information from the original schematic (as opposed to a timing netlist generated by the Xilinx software) to model the design.

4. Click **OK** to start QuickSim II.

Viewing the Calc Schematic

When QuickSim starts, no windows are open. In this section, you 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.

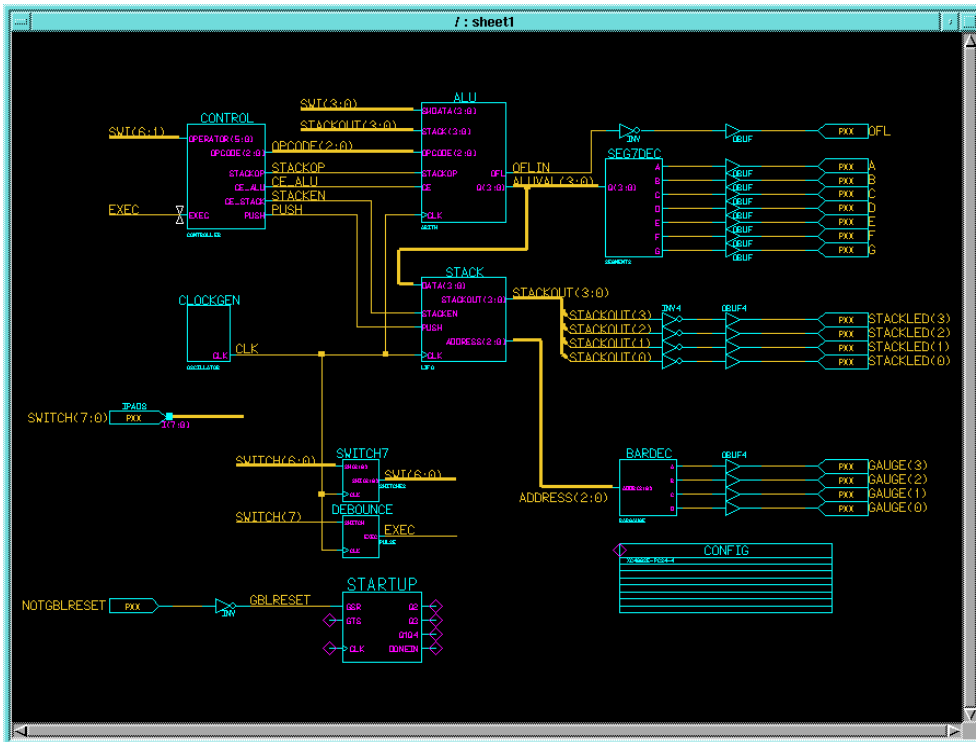


Figure 9-55 Top-Level Calc Schematic

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 follow these steps:

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.

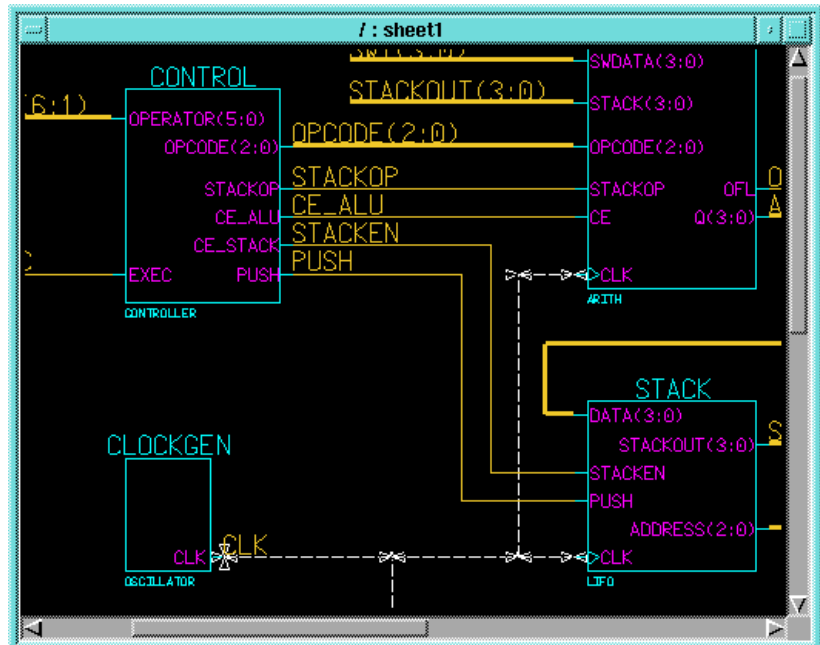


Figure 9-56 Selecting the CLK Net for Display in Trace Window

3. Use the left mouse button to select the following nets: STACKEN, PUSH.
4. Using the **Shift-F8** key, view the entire schematic.
5. Select the net labeled EXEC (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 ALUVAL(3:0) and STACKOUT(3:0).
7. Press the blue **TRACE** button in the palette to add all selected signals to the Trace window.

Opening Trace and List Windows

To view the waveforms of the selected signals, you must open a QuickSim Trace window.

To open a Trace window, perform these steps.

1. Select the blue button labeled **TRACE** in the palette with the left mouse button.

A Trace window appears displaying the waveforms selected on the schematic.

2. If necessary 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 through 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 is highlighted, and the related component on the schematic page also appears highlighted.

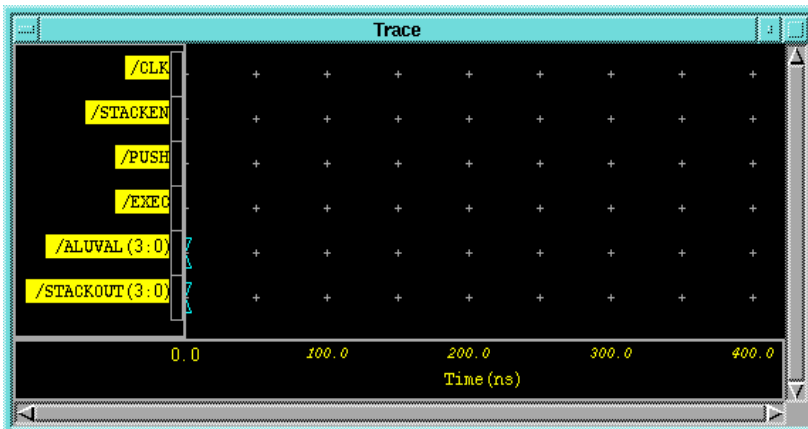


Figure 9-57 Trace Window

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 these steps.

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.

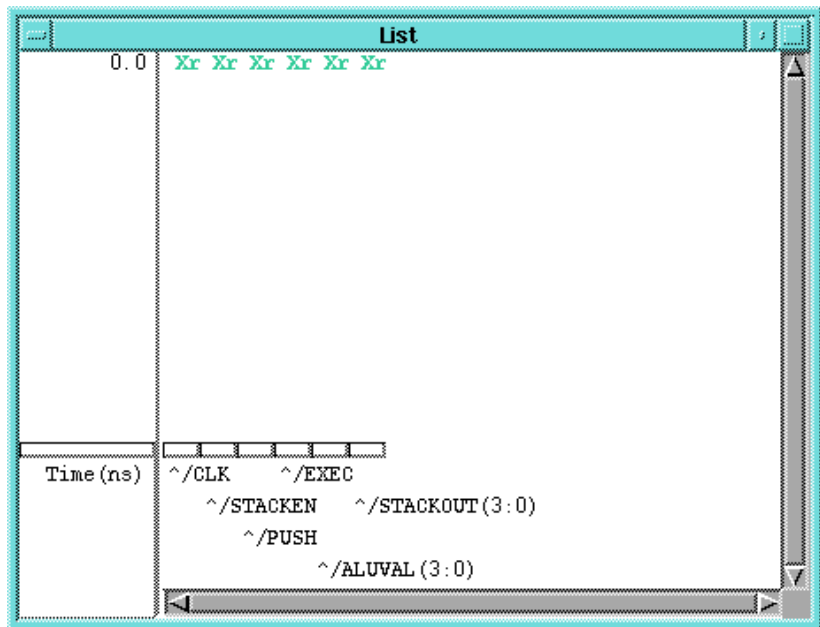


Figure 9-58 List Window

Adding Traces Manually

In the Calc design, inputs are entered via a set of eight switches, SWITCH(7:0). The lower seven switches (SWITCH(6:0)) define the opcode. The left-most switch (SWITCH(7)) is the execute switch.

When SWITCH(7) is toggled, the selected opcode on SWITCH(6:0) is executed. It is useful to view SWITCH(6:0) and SWITCH(7) separately in the Trace window.

Add these two traces to the Trace window as follows:

1. To add traces manually, the Stimulus Palette must be active. Click on the red **STIMULUS** button in the palette. The icons in the palette change.
2. Press the **F2** key to unselect everything.
3. Select the Trace window with the left mouse button.
4. Choose **Right Mouse Button** → **Add** → **Traces** → **Specified**.
5. In the dialog box that appears, select the Named Signals button with the left mouse button.
6. Fill in the dialog box as shown in the figure below.
7. Select **OK** or press Return.

The bus SWITCH(6:0) and the signal SWITCH(7) are added to the Trace window.

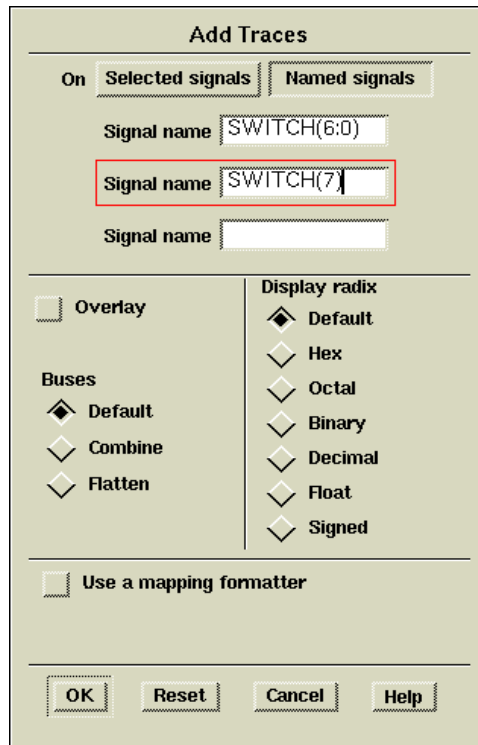


Figure 9-59 Adding Traces Manually

Assigning Values to the Clock

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 **ADD CLOCK** in the palette.
4. Fill in the dialog box that appears as shown in the following figure.

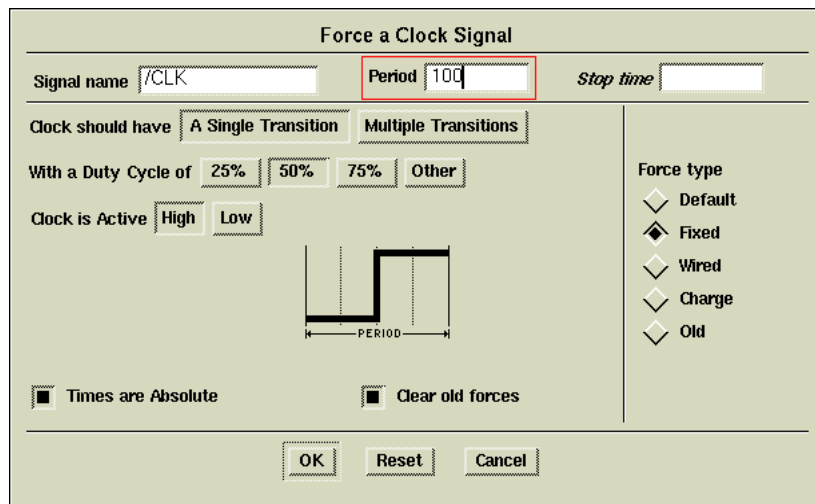


Figure 9-60 Adding a Clock Waveform

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 nanoseconds. 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.

5. Press return or select **OK** to add the force to CLK.

Asserting Global Set/Reset (without STARTUP)

Note: This section applies to designs in which the STARTUP module has not been instantiated. If you have an XC4000 family or XC5200 design that has the STARTUP module in it, go to the “[Asserting Global Set/Reset \(with STARTUP\)](#)” section.

In every simulation, the first node you must assert is the global-reset signal. This signal does not exist on your schematic, but does exist in the device. This dedicated net is connected to every asynchronous reset pin on every flip-flop (including IOB flip-flops) in an FPGA or CPLD. The net is named differently and has a particular polarity depending on the device family used as shown in the following table.

Table 9-3 Net Names for the Global Set/Reset Signal

Device Family	Net Name	Polarity
XC3000	//globalresetb	Active Low
XC4000	//globalsetreset	Active High
XC5200	//globalreset	Active High
XC7000	//prld	Active High
XC9000	//prld	Active High

In each case, the global-reset net name is preceded by two forward slashes, indicating that the net is a global signal in QuickSim. The global-reset signal 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.

In the following example, you set //prld, the XC9000 global-reset signal, high at time 0 and low at 100 ns:

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.

The Force Multiple Values 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 signal is to be forced low (asserted) at time zero and high at time 100ns.

The image shows a dialog box titled "Force Multiple Values". It has a "Signal name" field containing "//prld". Below it are three rows of "Value" and "Time" input fields. The first row has Value "1" and Time "0". The second row has Value "0" and Time "100", with the "Time" field highlighted by a red rectangle. The third row has empty "Value" and "Time" fields. To the right is a "Force type" section with radio buttons for "Default", "Fixed", "Wired", "Charge", and "Old", and a checkbox for "Absolute". At the bottom are buttons for "OK", "Reset", "Cancel", and "Help".

Figure 9-61 Forcing the Global-Reset Signal (XC9000)

For other families, assert the appropriate signal with the appropriate polarity. For example, in the XC3000 family, the global-reset signal is //globalresetb, which must be forced low at time 0, and so on.

Asserting Global Set/Reset (with STARTUP)

Note: This section applies to designs in which the STARTUP module has been instantiated. If you have a design without a STARTUP module instantiated, follow the instructions in the [“Asserting Global Set/Reset \(without STARTUP\)”](#) section.

The global-reset signal must be forced at the beginning of all XC4000 family and XC5200 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 FDP or an FDC 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 other families, the global-reset signal in the XC4000 family and XC5200 family is not hard-wired to a package pin, and need not appear on one at all. If you want access to the global-reset net from an external pin, place the STARTUP component in your schematic and attach an IPAD and IBUF to the GSR pin for XC4000 family designs, or to the GR pin for XC5200 family designs. This pad becomes an

active-High Global Set Reset signal in XC4000 family devices and an active-High Global Reset signal in XC5200 family devices. You can also use an internally generated signal to drive the GSR or GR 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.

Since an external signal is connected to the global-reset net via the STARTUP symbol, you must pulse this external signal to activate global reset as opposed to the internal global-reset signal (explained in the “**Asserting Global Set/Reset (without STARTUP)**” section.

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. Normally, the net name //globalsetreset (XC4000) or //globalreset (XC5200) would be added as the signal name. (The two leading forward slashes would indicate that this is a global signal.) However, since you have included the STARTUP symbol in this design, you must instead pulse whatever signal is driving the GSR pin on the STARTUP module. In this case, pulse the NOTGBLRESET signal.
4. Fill in the dialog box as shown in the following figure. The NOTGBLRESET signal is to be forced low (asserted) at time zero and high at 100ns. Note that, because of the inverter in the path from NOTGBLRESET to GSR or GR, this series of forces is equivalent to pulsing globalsetreset or globalreset high.

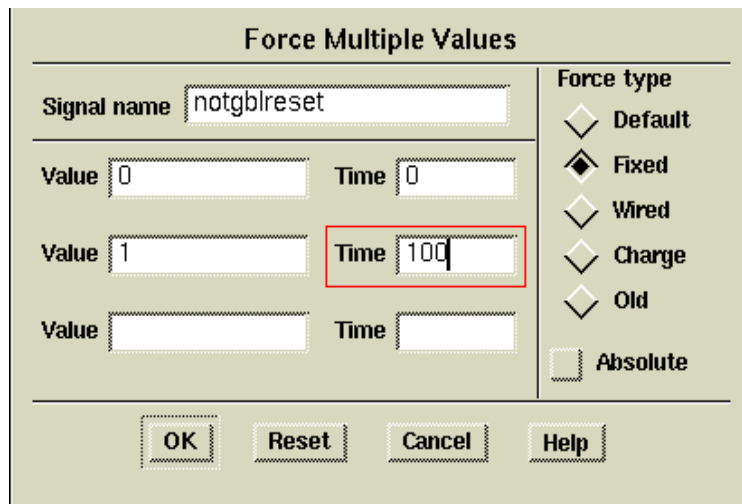


Figure 9-62 Forcing Globalsetreset via Notgblreset (XC4000E)

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 ALUVAL(3:0), the current contents of the internal register, and STACKOUT(3:0), the top value in the stack.

For a more detailed description of the Calc design, see the [“Design Description” section](#).

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 (SWITCH(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.

Force Multiple Values	
Signal name	switch(6:0)
Value	00
Time	0
Value	61
Time	200
Value	0D
Time	700
Value	7B
Time	1200
Value	3F
Time	1800
Value	7B
Time	2300
Value	50
Time	2800
Value	
Time	

Force type

Default

Fixed

Wired

Charge

Old

Absolute

OK Reset Cancel Help

Figure 9-63 Forcing Values to SWITCH(6:0)

For these commands to be executed, you must provide stimulus to SWITCH(7), the execute switch. Perform the following actions to force SWITCH(7) correctly:

1. Press the blue **Unselect All** button in the palette.
2. Select the **SWITCH(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@@/SWITCH(7)`. While the `SWITCH(7)` trace represents the value of `SWITCH(7)` up to the present time in simulation, the trace `forces@@/SWITCH(7)` represents all values that will ever be forced on the signal. During simulation, this waveform can be edited to modify future values of `SWITCH(7)`.

A blue line appears extending from `SWITCH(7)` to indicate that it has not been given a value. First, force `SWITCH(7)` to a known value at time zero as follows:

1. Select the **ADD** 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@@/SWITCH(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 click **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.

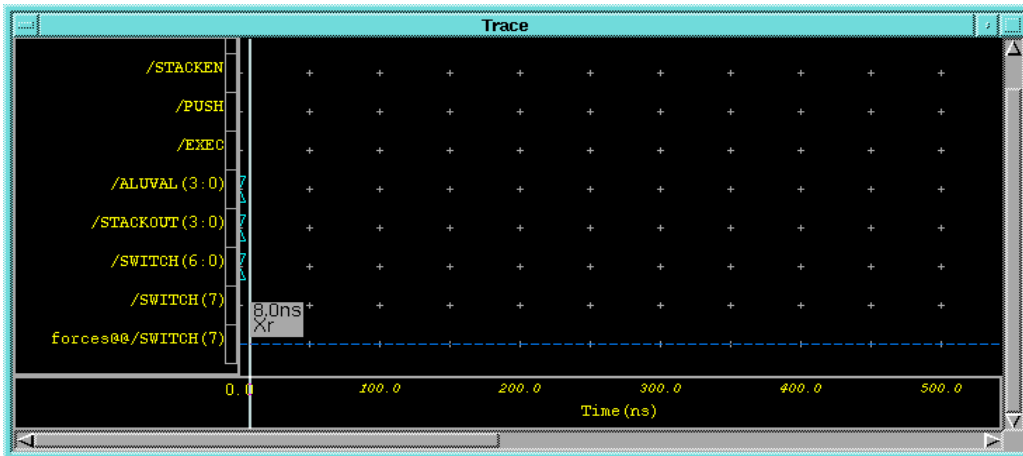


Figure 9-64 Forcing SWITCH(7) to Initial Value

5. Press the **Escape** key to end the Add Event operation.

Now that SWITCH(7) has been given an initial value, you must define when transitions occur on the signal as follows:

1. Select the **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@@/SWITCH(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 700 ns.

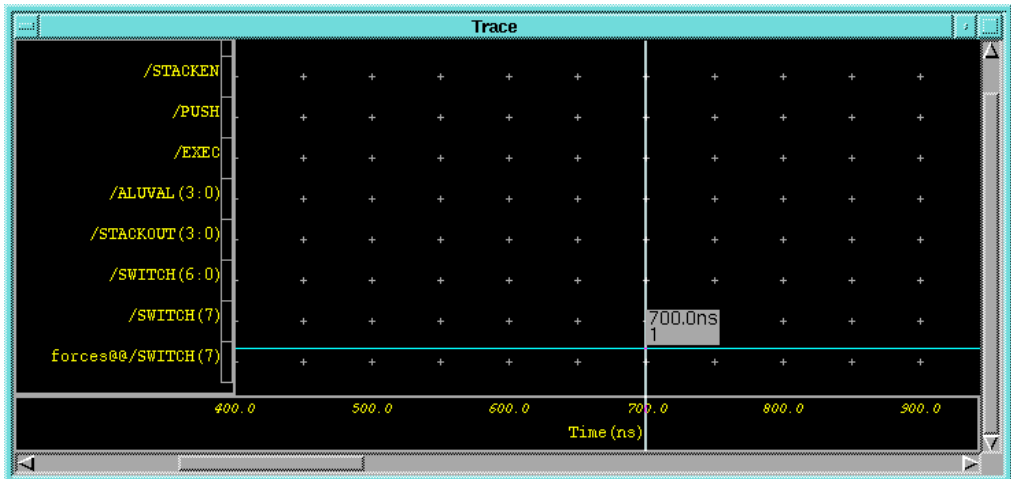


Figure 9-65 Adding the First Toggle to SWITCH(7)

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 icon in the palette and click the left mouse button on the incorrectly placed edge. The edge disappears. Then, select 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 TOGGLE command

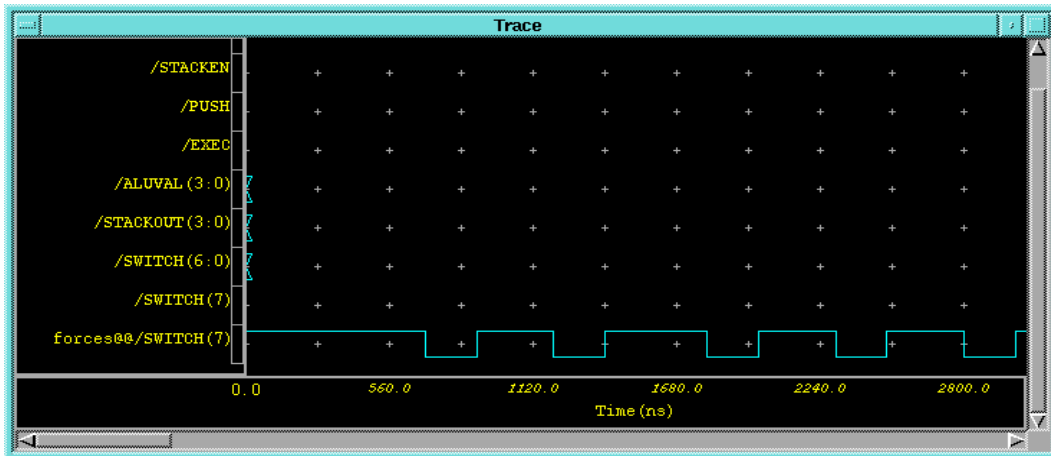


Figure 9-66 SWITCH(7) Force Waveform

Now that your inputs and clock are defined, you are ready to run the simulation.

6. 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.

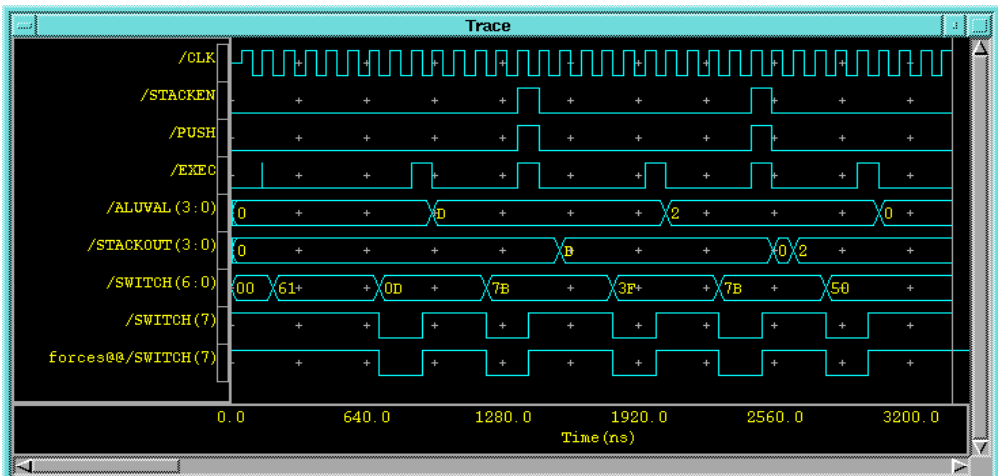


Figure 9-67 Output from Simulation (XC4000E design)

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.

The Save Waveform DB dialog box appears.

3. Fill in the Save Waveform DB 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 click **OK**.

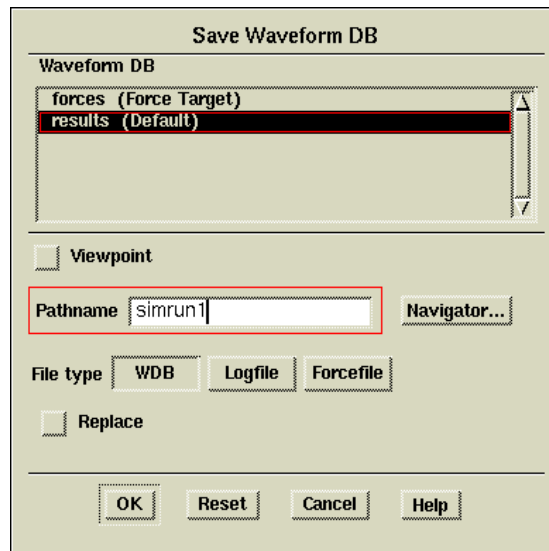


Figure 9-68 Saving Results

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.
The Save Waveform DB dialog box appears.
3. Fill in the Save Waveform DB 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 click **OK** to save the forces.

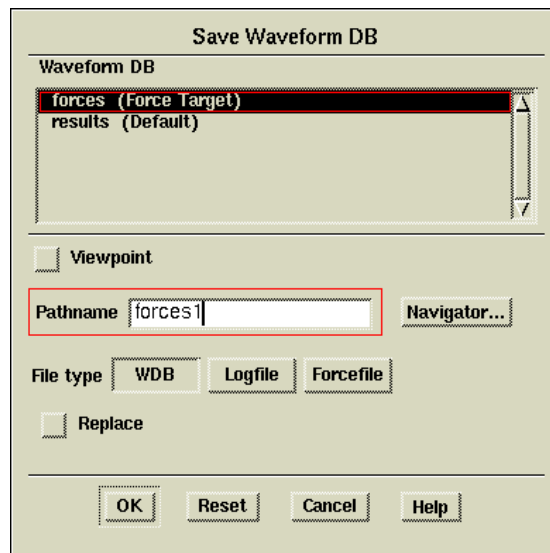


Figure 9-69 Saving Forces

After saving the results, reset the simulator to time zero as follows:

1. Press the blue **RESET** button in the palette.
2. In the dialog box that appears, select the **state** button so that it highlights, and 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.

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 (Advanced Multi-Purpose Language) is a C-like programming language used by all of the Mentor Graphics tools.

2. Select **Right Mouse Button → Export**.
3. In the dialog box that appears, 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 the appropriate Mentor Graphics documentation.

4. Select **File → Quit** to exit QuickSim.

Using Pld_men2edif

Once your design is verified to be functionally correct, you use `pld_men2edif`, a tool in the Mentor Graphics Design Manager, to translate your Mentor design into a Xilinx-ready EDIF netlist. Running `pld_men2edif` is always the first step in implementing a

design. Whenever you make changes to your schematic, you must run `pld_men2edif` again so that the Xilinx software can process those changes.

When you run `pld_men2edif` from the Mentor Design Manager, the following dialog box appears:

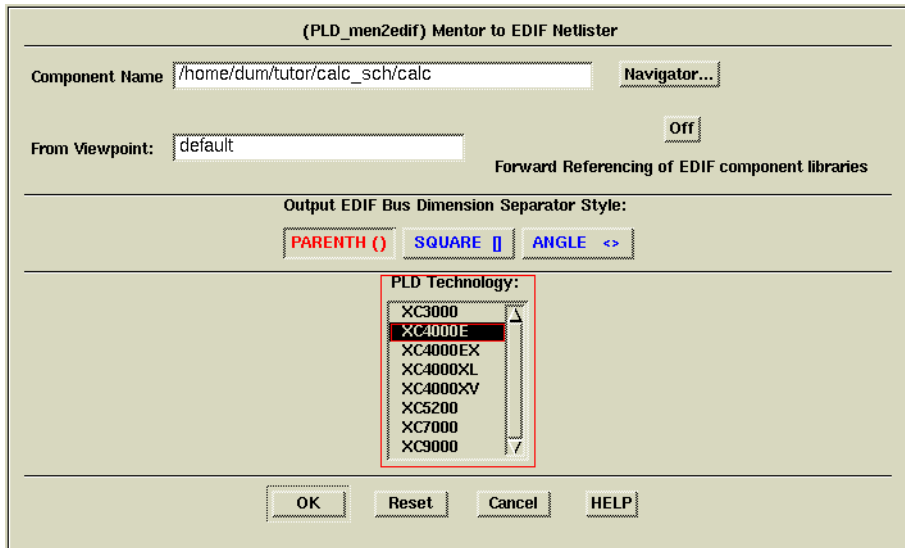


Figure 9-70 Pld_men2edif Dialog Box

Here is an explanation of some of the fields and buttons in the `pld_men2edif` dialog box.

- **Component Name**—Enter the name of the component that you want to process here.
- **From Viewpoint**—If you are an advanced Mentor Graphics designer who uses viewpoints to organize design models and properties, enter the viewpoint name that you wish to use for this EDIF translation. If you do not use or are not familiar with viewpoints, leave this field blank and `pld_men2edif` will use a default value.
- **Forward Referencing of EDIF component libraries**—This option applies only in rare situations where design hierarchy has been structured in such a way that circular or recursive references exist. Normally, this option is set to Off.

- **Output EDIF Bus Dimension Separator Style**—This determines how bus-index delimiters are written into the output EDIF file. This is important if you are merging components from other design-entry tools into a single design. Choosing a bus-index delimiter lets you insure that the bus-index delimiters that `pld_men2edif` writes out are consistent with those of any other design-entry tools with which you are interfacing.

Since this design has been fully captured in Mentor Graphics, you need not worry about what type of bus delimiters are used. You may leave this setting on the default (PARENTH).

- **PLD Technology**—Select the architectural family from this list.
- **HELP**—If the HELP button is clicked, a short help listing is produced by the `pld_men2edif` script.

Perform the following steps to create an EDIF netlist for Calc:

1. Double-click on the `pld_men2edif` icon in Design Manager.
2. For the Component Name, type `$XILINX_TUTORIAL/calc_sch/calc` as shown above.
3. Select the architecture in the PLD Technology field, *e.g.*, XC4000E.
4. Select **OK**.

This opens a new shell window where `pld_men2edif` runs and reports its progress. When `pld_men2edif` has completed, the following should appear at the bottom of the shell window:

```
pld_men2edif ended with return code 0
Done.
```

5. Dismiss the `pld_men2edif` shell window by typing `Ctrl-C` in it or by selecting **Close** from the window's control menu (accessed through the button on the left side of the title bar).

Note: The output of `pld_men2edif` may be sent to the window from which the `pld_dmgr` was originally invoked. This behavior is dictated by the `$MGC_TERMINAL_WINDOW` environment variable; see the Mentor Graphics documentation for more details.

Examining Pld_men2edif Output Files

In addition to the EDIF netlist, `pld_men2edif` also creates a `pld_men2edif.log` file. This file contains a transcript of the processing

done by `pld_men2edif`. If the program fails to generate an EDIF netlist, any errors encountered are logged in this file.

Examine the `pld_men2edif.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_men2edif`, including an EDIF file for Calc, and a log file for `pld_men2edif`.

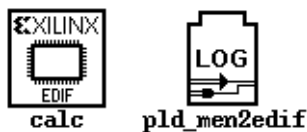


Figure 9-71 Files Created by Pld_men2edif

3. Select the LOG icon labeled `pld_men2edif` and choose **Right Mouse Button** → **Open** → **Editor**.

A window appears displaying the log file.

4. When you are done viewing the log, close the window.

Note: You can change the display font in this window by selecting **View** → **Fonts**.

Using the Xilinx Design Manager

The Xilinx Design Manager is a graphical design-flow and project manager. The Xilinx Design Manager takes your design, represented by the EDIF file from `pld_men2edif`, and implements it in an FPGA or CPLD. You can also use the Xilinx Design Manager to generate timing information that you can import into QuickSim or QuickHDL.

This section gives a brief overview of the design implementation flow. For a more in-depth discussion of the flow, including advanced implementation options, see the *Development System Reference Guide*.

1. Within the Mentor Design Manager, select the **Calc EDIF** icon in the Navigator, then select **Right Mouse Button** → **Open** → **pld_dsgnmgr**.

The Xilinx Design Manager appears as shown. The tool automatically creates a Xilinx project called calc. Xilinx project information is kept in a file called xproject/calc.prj by default.

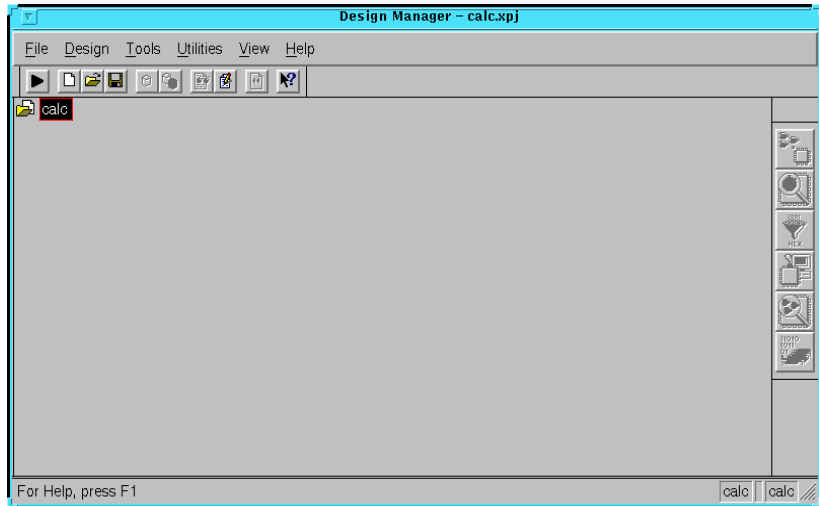


Figure 9-72 Xilinx Design Manager

Each project has associated with it objects known as “versions” and “revisions.” Versions represent logic changes in a design (for example, adding a new block of logic, replacing an AND gate with an OR gate, or adding a flip-flop); revisions represent different executions of the design flow on a single design version, usually with new implementation options (for example, higher place and route effort, a change in part type, or experimentation with new bitstream options). In the next stage, you make a new version and revision on which you run the implementation design flow.

2. Within the Xilinx Design Manager, select **Design** → **Implement**, which gives you the Implement dialog box, with fields for part type, design version, and revision as shown in the following figure.

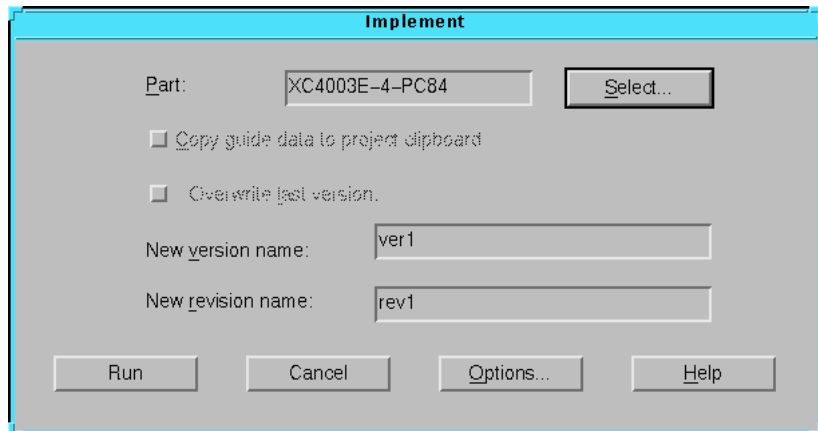


Figure 9-73 Implementation Dialog Box

In the current release of software, the Xilinx Design Manager does not read the part type from the design.

Note: The PART property in the CONFIG symbol does not get read properly when processed from the system prompt, if the “-p” command-line option is omitted from NGDBUILD and MAP. (See the **“Command Summaries”** section of this chapter.)

3. Click the **select** button to display a pull-down listing of available devices.
4. Choose a Family of **XC4000E**, a Device of **XC4003E**, a Package of **PC84**, and a Speed Grade of **-4**.
5. Click **OK**.

The part number is inserted into the Part field in the Implement dialog box.

6. Click on **Options**.

The Options dialog box appears.

Note: The CPLD Options dialog box does not have a Configuration Template section, nor does it have a Produce Logic Level Timing Report checkbox.

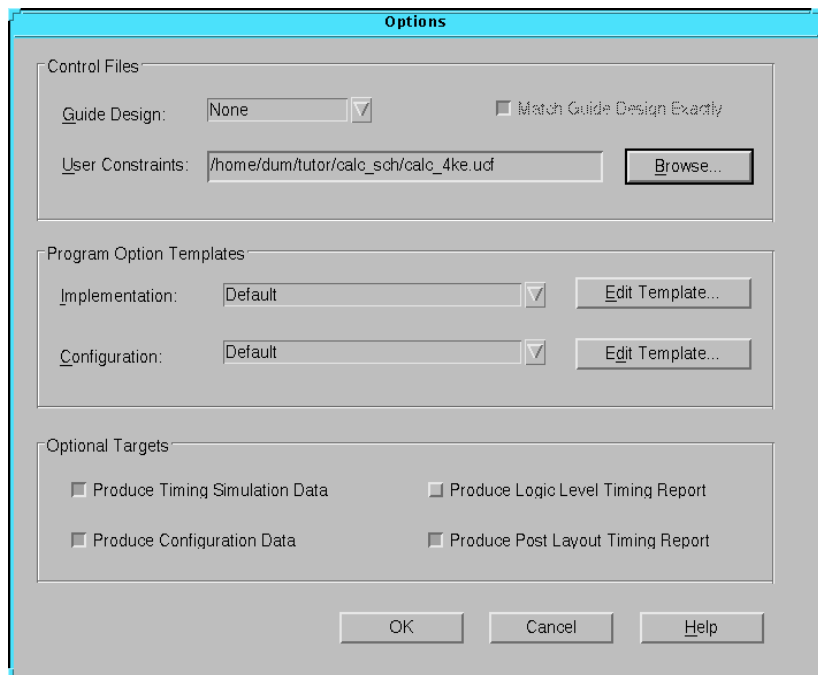


Figure 9-74 Options Dialog Box

7. Click **Browse** by the User Constraints field.
8. Select the **calc_4ke.ucf** file from the design directory, then Click **OK**.
9. Under Optional Targets, make sure the following are selected:
 - **Produce Timing Simulation Data**—This generates a back-annotated EDIF netlist that can be imported into the Mentor Graphics tools.
 - **Produce Configuration Data**—This generates a programming bitstream suitable for downloading into the Xilinx device.
 - **Produce Post Layout Timing Report**—This generates a timing report file based on how the design is actually routed.

You can also select the following option (FPGAs only):

- **Produce Logic Level Timing Report**—This generates a preliminary (post-map, pre-place and route) timing report based on the number of logic levels in each signal path. Since it is generated after the mapping step but before the place-and-route layout step, it does not contain information on device routing. Looking at this report before place and route can be useful for seeing how much “routing slack” you have in a design.
10. Under Program Option Templates Implementation, select **Edit Template**.

The XC4000 Implementation Options dialog box appears as shown.

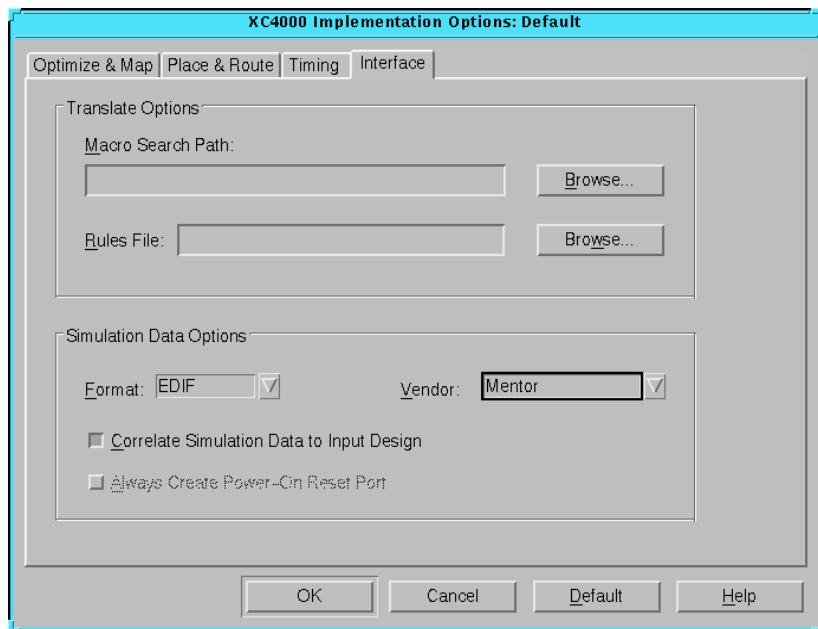


Figure 9-75 Changing EDIF Vendor Information

11. Select the **Interface** tab.
12. In the Interface pane, look under Simulation Data Options and verify that Format is set to **EDIF** and that **Correlate Simulation Data to Input Design** is selected.
13. In the Vendor field, select **Mentor**.

14. Click **OK** to return to the Options window.
15. Click **OK** to return to the Implementation dialog box.
16. Verify that the version is “ver1” and the revision is “rev1” then click **Run**.

The Flow Engine comes up as shown in the figure.

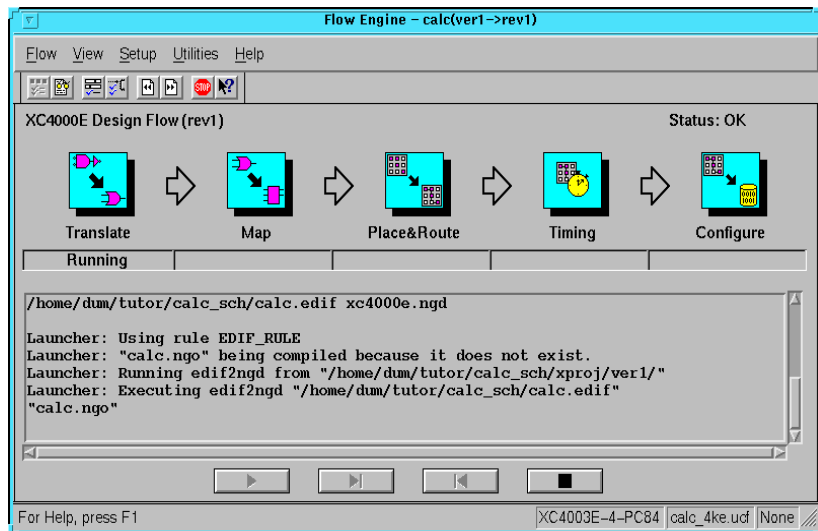


Figure 9-76 The Xilinx Flow Engine

The status bar shows the progress of the implementation flow with the following stages:

- **Translate**—convert the design EDIF file into an NGD (Native Generic Design) file
- **Map**—group basic elements (“bels”) such as flip-flops and gates into logic blocks (“comps”); also generate a logic-level timing report if desired
- **Place&Route**—place comps into the device, and route signals between them
- **Timing**—generate timing simulation data and an optional post-layout timing report
- **Configure**—generate a bitstream suitable for downloading into and configuring a device

When the implementation completes, an Implementation Status box displays:

```
Implementing revision ver1->rev1 completed  
successfully.
```

17. Click on View Logfile to display the logfile from the Flow Engine.

The report is displayed in vi.

18. To exit the viewer, type `:q!` and press **Return**.
19. Click **OK** in the Implementation Status dialog to return to the Xilinx Design Manager.

Note: To use another text editor, such as Emacs, as the report viewer, select **File** → **Preferences** from the Xilinx Design Manager.

Performing Timing 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. Also, since the delay-annotated timing netlist is different from the original schematic design, the timing simulation uses a process called *cross-probing* to allow you to view simulation results on your schematic. In this section, you perform a timing simulation of the Calc design by first preparing the design using `pld_edif2tim`. Once this has been done, you run `pld_quicksim` with cross-probing to trace waveforms and annotate results onto your original schematic.

Using Pld_edif2tim to Prepare a Timing Simulation

`Pld_edif2tim` reads a routed EDN file and back-annotates the delays to the schematic. This includes a number of steps, all of which are automatically run by the `pld_edif2tim` script. This script is represented by the `pld_edif2tim` icon in `pld_dmgr`. The files necessary for back-annotation have either been created in the Design Architect tutorial or are included in the solution directories.

Use `pld_edif2tim` to prepare the design for timing simulation as follows:

1. In `pld_dmgr`, use the Navigator to find and select the EDN `time_sim` icon.

This represents the timing-annotated netlist generated by the Xilinx Design Manager.

Note: There may be two similar looking types of icons, one marked EDIF and the other marked EDN. An EDIF file represents a netlist translated from the original schematic, while an EDN file represents a netlist translated from a routed NCD file. Be sure you select an EDN file to prepare for timing simulation.

2. Select **Right Mouse Button** → **Open** → **p1d_edif2tim**.

A dialog box appears. Design Manager automatically fills in the dialog box with the name of the EDN file.

3. Verify that the Replace existing routed design library field is set to **NO**.

On subsequent executions of `p1d_edif2tim`, you may set this to **YES** if you are overwriting a previous timing model.

4. Press **return** or select **OK** to execute the command.

The script produces a shell and runs in it.

Examining the P1d_edif2tim.log File

Examine the `p1d_edif2tim.log` file as follows:

1. In `p1d_dmgr`, select **Right Mouse Button** → **Update Window**.

The window is updated with the files that `p1d_edif2tim` generated.

2. Find the `p1d_edif2tim` 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 `p1d_edif2tim` during the timing flow, see the **“Command Summaries”** section at the end of this chapter. The timing flow is always the same since the starting point is always a routed EDN file with delays.

4. When you have finished looking at the file, close the Editor window.

Using Pld_dve

As with functional simulation, the timing-annotated netlist must also have a viewpoint associated with it.

1. In your design directory, you should now see a directory called `calc_lib`. Double-click on this directory icon to descend into it.
2. Select the `calc` component (which should be at the very bottom on the icon listing) with the left mouse button.
3. Invoke `pld_dve` on the simulation netlist component by selecting **Right Mouse Button** → **Open** → `pld_dve`.

A dialog box appears. Note that the component name, `Calc`, is entered automatically with a fully qualified path.

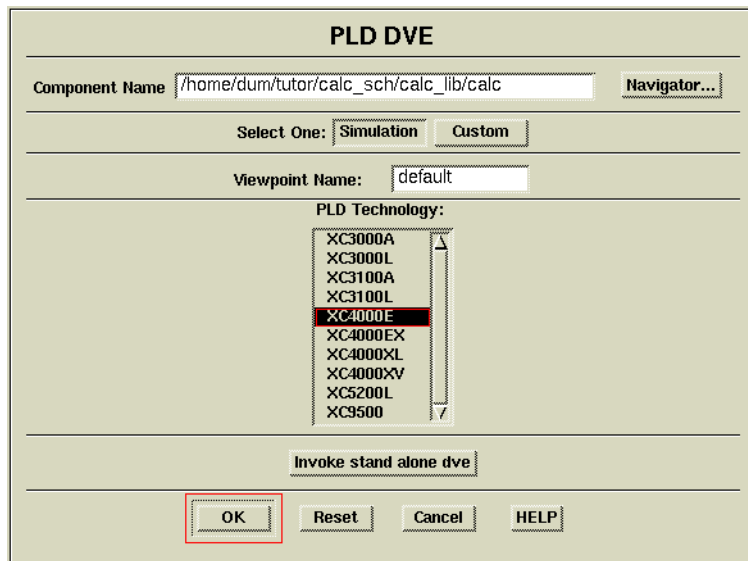


Figure 9-77 Invoking Pld_dve for Timing Simulation

4. Select the appropriate PLD Technology from the listing, *e.g.*, `XC4000E`, as shown in the figure above. (Leave other options set to their defaults, as shown in the figure.)
5. Click **OK** to execute the `pld_dve` script.
6. Once `pld_dve` completes, dismiss the shell window in which it has executed.

Invoking QuickSim for Timing Simulation

1. With the timing-annotated calc component in the calc_lib directory still selected, invoke pld_quicksim by selecting **Right Mouse Button** → **Open** → **pld_quicksim**.

A dialog box appears. The component name, Calc, is entered automatically with a fully qualified path.

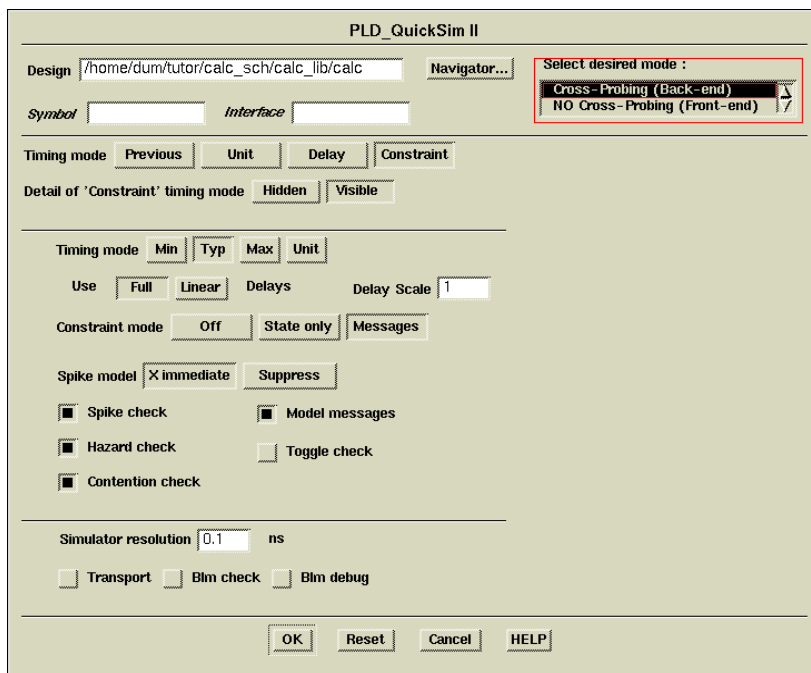


Figure 9-78 Invoking Pld_quicksim for Timing Simulation

2. Select desired mode as **Cross-Probing**.

This allows QuickSim to use the back-annotated timing model in QuickSim, while allowing you to view the original schematic in DVE. This process is necessary because your original schematic is expressed in Unified libraries, while the back-annotated timing model is generated using simulation primitives.

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.

The Design Viewpoint Editor (DVE) appears and gives an informational message reading,

```
To start the cross-probing process, ...
```

8. Click **Close** in this message window.
9. Resize the DVE window so that it is almost as large as the entire screen.

The QuickSim window also appears.

10. Resize the QuickSim window so that it is almost as large as the entire screen, allowing space for you to click on the DVE window to make it active and display in the foreground.
11. Bring the DVE window to the foreground and select **OPEN DESIGN VIEWPOINT** from the palette.
12. In the dialog box, enter the name of the *original* component in the Component field, e.g., `$XILINX_TUTORIAL/calc_sch/calc`. (Do *not* enter the name of the simulation model, `$XILINX_TUTORIAL/calc_sch/calc_lib/calc`.)
13. Click **OK** to load the original viewpoint.
14. Select **OPEN SHEET** from the palette to open the top-level Calc schematic.
15. To see the cross-probing process in action, click on the **CLK** net attached to the **CLOCKGEN** component.

A few seconds after this net is selected in DVE, a Trace window appears in QuickSim listing the CLK net.

16. Bring the QuickSim window to the foreground to see the Trace window.
17. Select **Transcript** → **Replay** from the QuickSim menu bar.
18. From the dialog box, choose the `calc_4ke.do` file. (Depending on your target device, select `calc_3ka.do`, `calc_5k.do`, or `calc_9k.do`.)

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.

The figure below shows how DVE and QuickSim may look like running side by side.

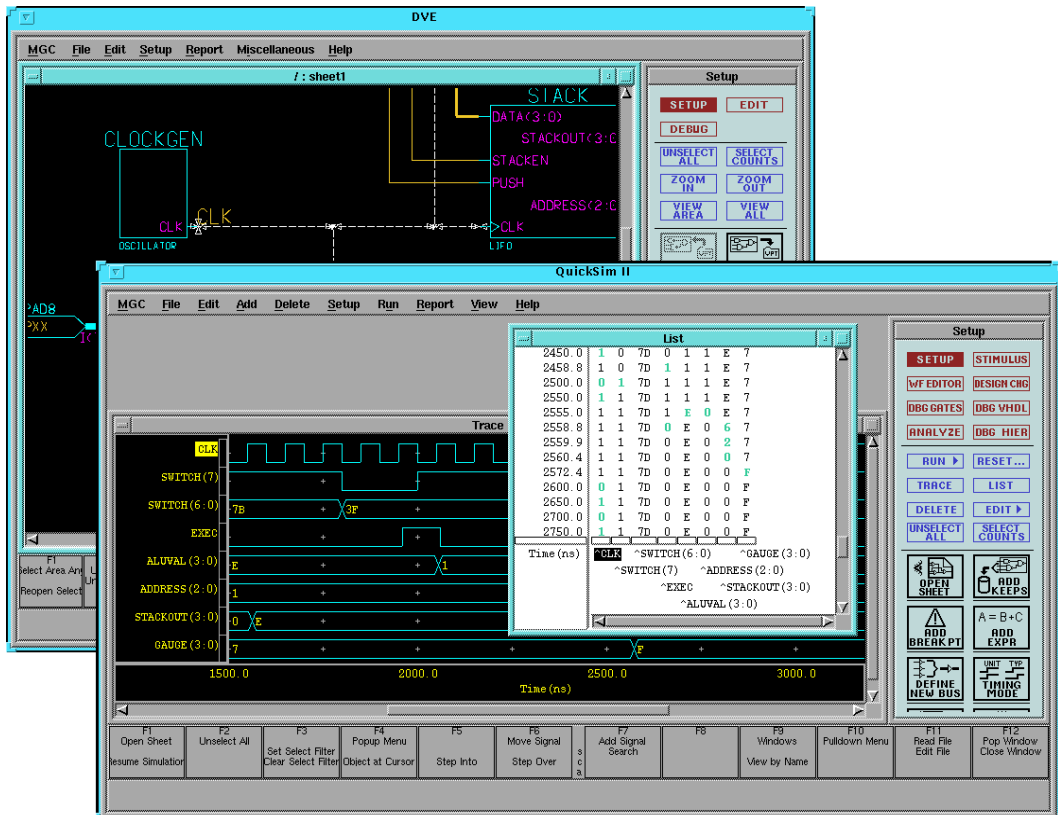


Figure 9-79 Cross-Probing with DVE and QuickSim

19. After examining the waveforms in timing simulation, close both the QuickSim and DVE windows.

Examining Routed Designs with EPIC

Note: This section applies only to FPGA designs. If you are targeting a CPLD such as an XC9000 device, skip to the “[Making Incremental Design Changes](#)” section.

At this point in the tutorial, the design process is complete. If you would like to see how the design has been implemented by the Xilinx software, you can take a graphic look at your placed and routed design using the Editor for Programmable Integrated Circuits, or EPIC. You can access EPIC from the Xilinx Design Manager.

EPIC provides several useful functions, such as:

- Manual placement of a pre-routed design
- Manual editing of a routed design
- Static timing analysis



Figure 9-80 EPIC Icon

EPIC is explained in a separate tutorial. See the “EPIC Tutorial” section of the *EPIC Reference/User Guide*. Before starting this tutorial, be sure to select the `ver1` → `rev1` revision of the design in the project view

Verifying the Design Using a Demonstration Board

Note: This section applies only to FPGA designs. If you are targeting a CPLD such as an XC9000 device, skip to the “[Making Incremental Design Changes](#)” section

Creating and Downloading the Bitstream

A bitstream has been created during the Configure stage in Flow Engine. At this point, you are ready to download the bitstream using a parallel download cable or the more versatile XChecker cable connected to your workstation. The XC4000E version of the Calc design is suitable for download into an FPGA demonstration board available from Xilinx.

Downloading is accomplished with Hardware Debugger. To invoke Hardware Debugger, you select **Tools** → **Hardware Debugger** from the menu bar, or click the Hardware Debugger icon on the toolbar. If you are using an XChecker cable, you can also use the Hardware Debugger to read back information from the device to verify both the configuration as well as the state of memories and registers within the device.



Figure 9-81 Hardware Debugger Icon

Hardware Debugger is explained in a separate tutorial. See the “CALC Tutorial” section of the *Hardware Debugger Reference/User Guide*. Before starting this tutorial, be sure to select the `ver1` → `rev1` revision of the design in the project view.

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,” since much of it is unchanged. This process is known as incremental design, and the NCD file (containing partition, placement, and routing information) from the prior place and route run is used as a guide.

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 passes 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 options available in the Xilinx Flow Engine.

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, especially to existing portions of the design, it can be disadvantageous to guide the design.

Making an Incremental Schematic Change

Make a simple change to the Calc schematic that is 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 grounding the 'R' pins that are inputs to the FDRE and FD4RE macros in the ALU schematic. The logic that generated the original reset signal, and the logic it drove, is automatically optimized out of the netlist by the MAP program.

Open `p1d_da` and load the Calc schematic as follows.

1. From `p1d_dmgr`, select the `$XILINX_TUTORIAL/calc_sch/alu` design object and choose **Right Mouse Button** → **Open** → `p1d_da`.

Design Architect appears with the ALU schematic loaded.

2. Use the **F8** key, or the stroke 159, to zoom in on the lower right quadrant of the schematic.
3. Press the **F2** key to make sure nothing is selected.
4. Select the **AND5B2** component that generates the QRESET net feeding the FDRE and FD4RE.
5. Press the **Delete** key to delete the component.
6. Connect a ground symbol to the dangling QRESET net.

The GND symbol can be found in the **BY TYPE** → **general** section of the Xilinx Library menu. See the figure below.

7. Check and save the schematic.
8. Exit `p1d_da` and return to `p1d_dmgr`.

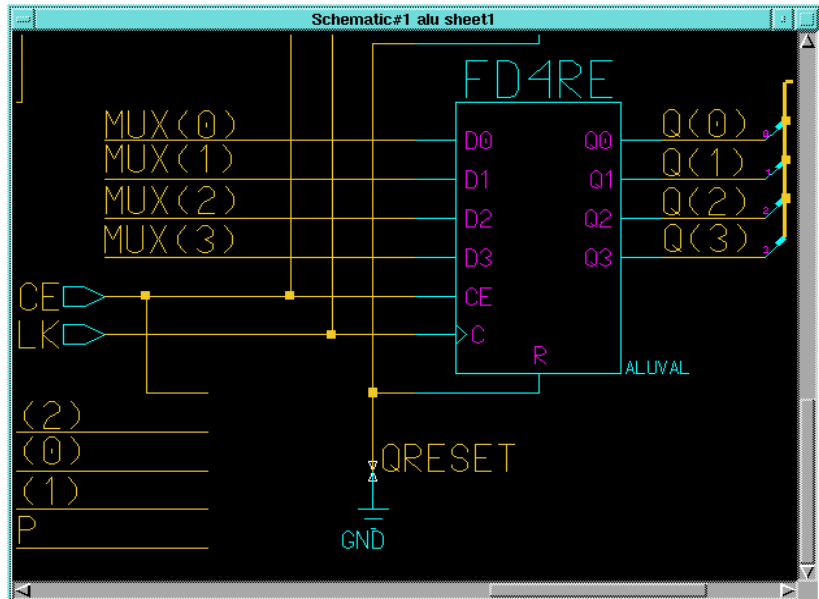


Figure 9-82 Grounding the Reset Logic

Translating the Incremental Design

Translate the guided Calc design by turning on the guide options in Flow Engine. The following instructions demonstrate an alternative method of running Flow Engine that offers more control over the implementation flow.

1. In the Xilinx Design Manager, select **calc**, then choose **Design** → **New Version**.
2. The New Version dialog box appears with the Name field automatically filled in as ver2. You may also add a comment to the new version. This comment appears in the project view next to the version number. Click **OK**.

Note: You can add a comment to any version or revision in the project view by selecting that version or revision, then selecting **Right Mouse Button** → **Properties**.

3. Select the newly created ver2 in the project view, then select **Design** → **New Revision**.

The New Revision dialog box appears with the Name field automatically filled in as **rev1** and the Part field automatically filled in as **XC4003E-4-PC84**. You may add a comment to the new revision if you wish.

4. Click **OK** to close the dialog box.
5. Select the newly created “rev1” in the project view, then select **Tools** → **Flow Engine**. Alternatively, you can click the Flow Engine icon in the Toolbox.

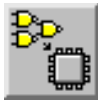


Figure 9-83 Flow Engine Icon

Flow Engine appears. However, unlike the procedure you used in the first revision, the implementation flow does not start automatically. This allows you to step forward and even backward through the implementation flow by individual stages, using the audio-player-like buttons at the bottom of the Flow Engine window, or the selections underneath the Flow menu.

6. Select **Setup** → **Options** from the menu bar.
The Options dialog box appears as before.
7. Go through the different options as before and verify that the settings you gave in the previous revision have been carried over into this revision.
8. In the Guide Design field, select Last.
This sets the previous revision of the placed and routed design. In this case, it has the same effect as selecting **ver1** → **rev1**.
9. Click **OK** to return to Flow Engine.
10. Run the implementation as before by clicking the **play** button (on the far left) at the bottom on the Flow Engine window.
11. When all steps have completed successfully, select **Flow** → **Close** to exit Flow Engine.

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. As before, see the “CALC Tutorial” section of the *Hardware Debugger Reference/User Guide* for more information. Before running through this tutorial, make sure that the `ver2` → `rev1` revision is selected in the project view.

Command Summaries

Although this tutorial uses the Mentor Graphics Design Manager and the Xilinx Design Manager to process the Calc design, you can also manually run the individual programs that these graphical tools run.

This section details command sequences that you can use to perform the translations the Xilinx Design Manager performs in this tutorial. The commands are written as you would type them at the system prompt or in a batch file. You may also see a summary of these system commands by using the **Utilities** → **Command History** and **Utilities** → **Command Preview** selections in Design Manager or Flow Engine. Once you are in the Command History or Command Preview dialog box, select the display Mode to Command Line to see the detailed system commands, including command-line options, used by Flow Engine. You can cut and paste from these Command dialog boxes into your text editor to create batch files.

Note: The commands listed here are slightly different from the commands in the Command History and Command Preview windows. The commands listed here show how you would typically execute the Xilinx programs from the system prompt, outside of the Xilinx Design Manager framework. The commands listed in the Command History and Command Preview windows reflect how Flow Engine executes Xilinx programs to fit into the Xilinx Design Manager framework.

XC4000E Command Summaries

Functional Simulation

```
pld_dve -s calc xc4000e
pld_quicksim calc
```

Basic Translation

```
pld_men2edif calc xc4000e
ngdbuild -p XC4000E -uc calc_4ke.ucf calc.edif
calc.ngd
map -p XC4003E-4-PC84 -o calc_map.ncd -oe normal
calc.ngd calc.pcf
trce calc_map.ncd -a -o calc_map.twr
par -w -l 4 calc_map.ncd calc.ncd calc.pcf
trce calc.ncd -a -o calc.twr
ngdanno calc.ncd calc_map.ngm
ngd2edif -v mentor -w calc.nga calc.edn
bitgen calc.ncd -l -w
```

Timing Simulation

```
pld_edif2tim calc.edn
pld_dve -s calc_lib/calc xc4000e
pld_quicksim calc_lib/calc -cp -tim typ -consm
messages
```

Incremental Translation

```
mv calc.ncd calc_guide.ncd
mv calc_map.mdf calc_guide.mdf
pld_men2edif calc xc4000e
ngdbuild -p XC4000E -uc calc_4ke.ucf calc.edif
calc.ngd
map -p XC4003E-4-PC84 -o calc_map.ncd -oe normal
-gf calc_guide.ncd calc.ngd calc.pcf
trce calc_map.ncd -a -o calc_map.twr
par -w -l 4 -gf calc_guide.ncd calc_map.ncd
calc.ncd calc.pcf
trce calc.ncd -a -o calc.twr
ngdanno calc.ncd calc_map.ngm
ngd2edif -v mentor -w calc.nga calc.edn
bitgen calc.ncd -l -w
```

XC9000 Command Summaries

Functional Simulation

```
pld_dve -s calc xc9000
pld_quicksim calc
```

Basic Translation

```
pld_men2edif calc xc9000  
cpld -p XC95108-10-PC84 calc
```

Timing Simulation

```
ngd2edif -v mentor -w calc.nga calc.edn  
pld_edif2tim calc.edn  
pld_dve -s calc_lib/calc xc9000  
pld_quicksim calc_lib/calc -cp -tim typ -consm  
messages
```

Incremental Translation

```
cpld -p XC95108-10-PC84 -pinlock calc
```

Further Reading

The Schematic Design Tutorial is provided to give you the information necessary to begin a Xilinx design using Mentor Graphics software. It is important to note that tools as broad and complex as Design Architect and QuickSim cannot be fully explained in a single tutorial. There are many different ways to use the commands in these tools, and there are also many ways to customize these applications. It is strongly recommended that you read the Mentor Graphics Design Architect and QuickSim documentation as well as the *Xilinx Mentor Graphics Interface/Tutorial Guide*.

Schematic-on-Top with VHDL Tutorial

This chapter contains the following sections:

- “Introduction” section
- “Required Background Knowledge” section
- “Design Flow” section
- “Software Installation” section
- “Starting the Design Manager” section
- “Copying the Tutorial Files” section
- “Starting Design Architect” section
- “Completing the Calc Design” section
- “Using a Constraints File” section
- “Performing Functional Simulation” section
- “Using Pld_men2edif” section
- “Using the Xilinx Design Manager” section
- “Performing Timing Simulation” section
- “Examining Routed Designs with EPIC” section
- “Verifying the Design Using a Demonstration Board” section
- “Command Summaries” section
- “Further Reading” section

Introduction

This chapter guides you through a typical field-programmable gate array (FPGA) and complex programmable logic device (CPLD)

design procedure from schematic entry with instantiated HDL to completion of a functioning device. It uses a design called Calc, a 4-bit processor with a stack. In the first part of the tutorial, you use the Design Architect, the Mentor Graphics design entry tool, to link HDL entities to Mentor symbols and instantiate those symbols into the Calc design. Next, you use QuickHDL Pro, the Mentor Graphics schematic/HDL simulator, to perform a functional simulation on it. In the third step, you use the Xilinx Design Manager to implement the design. Finally, you verify the design's timing by using `pld_quicksim`. The simple design example used in this tutorial demonstrates many system features that you can apply to more complex FPGA and CPLD designs.

Note: Although this tutorial describes creating and processing FPGA designs, you can apply most of the steps to CPLD designs. Unlike the [“Schematic Design Tutorial” chapter](#), this tutorial focuses completely on FPGAs. For information on retargeting a design to a different device family, see the [“Targeting the Design for the XC9000 Family” section of the “Schematic Design Tutorial” chapter](#).

This tutorial includes instructions on the following:

- Installing the tutorial files
- Using Mentor Graphics Design Manager
- Using Mentor Graphics Design Architect
- Completing the SEG7DEC and ALU blocks in the Calc design
- Performing functional simulation on the Calc design in QuickHDL Pro
- Converting the design to an EDIF file using `pld_men2edif`
- Implementing the design using `pld_dsgnmgr`
- Configuring the Xilinx Design Manager/Flow Engine
- Performing timing simulation on the routed Calc design in `pld_quicksim`
- Examining routed designs with the Editor for Programmable ICs (EPIC)
- Verifying the Calc design on a demonstration board
- Command summaries

Required Background Knowledge

Note: This tutorial focuses specifically on the procedures for importing VHDL modules into Design Architect schematics. Refer to the “[Schematic Design Tutorial](#)” chapter for information on basic object-management procedures in Design Manager, schematic-entry procedures in Design Architect, or Xilinx-specific concepts such as CONFIG, STARTUP, and incremental design.

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

- UNIX operating system
- Motif Windows. Mentor Graphics applications conform to the Motif window style.
- Basic knowledge of Mentor Graphics tools, such as editing MGC location maps, manipulating design objects and launching applications in Design Manager, and adding and working with design elements in Design Architect. If you are not familiar with these procedures, see the “[Schematic Design Tutorial](#)” chapter.

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

Design Flow

See the “[Design Flows](#)” section of the “[Introduction](#)” chapter for the design flow involved in using the Mentor Graphics interface. That chapter also describes the general steps for creating a top-level schematic design with instantiated VHDL modules using the Mentor interface.

Note: To instantiate Verilog modules into a schematic in Design Architect, you must first encapsulate them within a VHDL entity. You then encapsulate the VHDL entity into the schematic.

The tutorial design is targeted for an XC4000E 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.

Software Installation

Required Software

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

- Mentor Graphics Version B.1 or later, including Mentor Design Manger, Design Architect, QuickSim, QuickPath, as well as the programs needed to read and write EDIF netlists (ENRead and ENWrite), which require special licensing
- A third-party FPGA/CPLD synthesis tool, such as FPGA Compiler (Synopsys), Synergy (Cadence Design Systems), or Galileo (Exemplar Logic)
- Xilinx/Mentor Graphics Interface Version M1
- Xilinx Development System Version M1

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 Version M1
- Xilinx/Mentor Graphics Interface Version M1
- Mentor Graphics Version B.1 or later, including Mentor Design Manger, Design Architect, QuickSim, QuickPath, as well as the programs needed to read and write EDIF netlists (ENRead and ENWrite), which require special licensing
- A third-party FPGA/CPLD synthesis tool, such as FPGA Compiler (Synopsys), Synergy (Cadence Design Systems), or Galileo (Exemplar Logic)

3. Verify the installation, using the **“Configuring Your System”** section of the **“Getting Started”** chapter as a guide.

4. Add a reference to `$XILINX_TUTORIAL` to your `MGC_LOCATION_MAP` file.

All of the tutorial designs use the variable `$XILINX_TUTORIAL` as part of their path references. For example, the design object `alu` in the `$XILINX/mentor/tutorial/calc_sot` directory uses the path reference `$XILINX_TUTORIAL/calc_sot/alu` to define where it is located in the directory structure.

With this definition added to the location map as defined in the **“Getting Started” chapter**, the complete location-map file should, at a minimum, look like:

```
MGC_LOCATION_MAP_1
(empty line)
$MGC_GENLIB
(empty line)
$LCA
(empty line)
$SIMPRIMS
(empty line)
$XILINX_TUTORIAL
/home/bclinton/mentor/xtutorial
```

This assumes the Xilinx tutorial files have been placed under `/home/bclinton/mentor/xtutorial`. For example, the full path to the schematic-on-top tutorial would be `/home/bclinton/mentor/xtutorial/calc_sot`

Refer to the **“Schematic Design Tutorial” chapter** or the Mentor Graphics documentation for more information on location maps.

Installing the Tutorial

The tutorial files are optionally installed when you install the Xilinx/Mentor Graphics interface software. If you have already installed the software, but are not sure whether you specified tutorial installation, check your software installation for a `$XILINX/mentor/tutorial` directory. This 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_name.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 contains 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 `$XILINX/mentor/tutorial` directory is created; design object directories are created; and the tutorial files needed to complete the design are copied to the `calc_sot` directory. Some of the files you need to complete the tutorial design are not copied, because you create these files in the tutorial. However, solutions directories with all input and output files are provided. They are located in the `$XILINX/mentor/tutorial` directory and are listed in the following table:

Table 10-1 Tutorial Design Directories

Directory	Description
<code>calc_sch</code>	Schematic (Design Architect) tutorial directory
<code>calc_4ke</code>	Schematic solution directory for XC4003E-PC84
<code>calc_9k</code>	Schematic solution directory for XC95108-PC84
<code>calc_sot</code>	Schematic-on-top tutorial directory (uses XC4003E)

The solution directories contain the design files for the completed tutorial, including schematics, intermediate directories, and the bitstream file. Different intermediate files are created for different device families. Do not overwrite any files in the solutions directories.

The `calc_sot` directory contains the incomplete copy of the tutorial design. The installation program copies a few intermediate files to the `calc_sot` tutorial directory, and you create the remaining files when you perform the tutorial. In a later step, you copy the `calc_sot` directory to another area and perform the tutorial in this new area. Each design component directory has associated with it a file called *file-name.mgc_component.attr*. This file identifies the corresponding directory as a Mentor Graphics design component.

Starting the Design Manager

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

```
p1d_dmgr
```

The Design Manager Window appears as shown in the following figure.

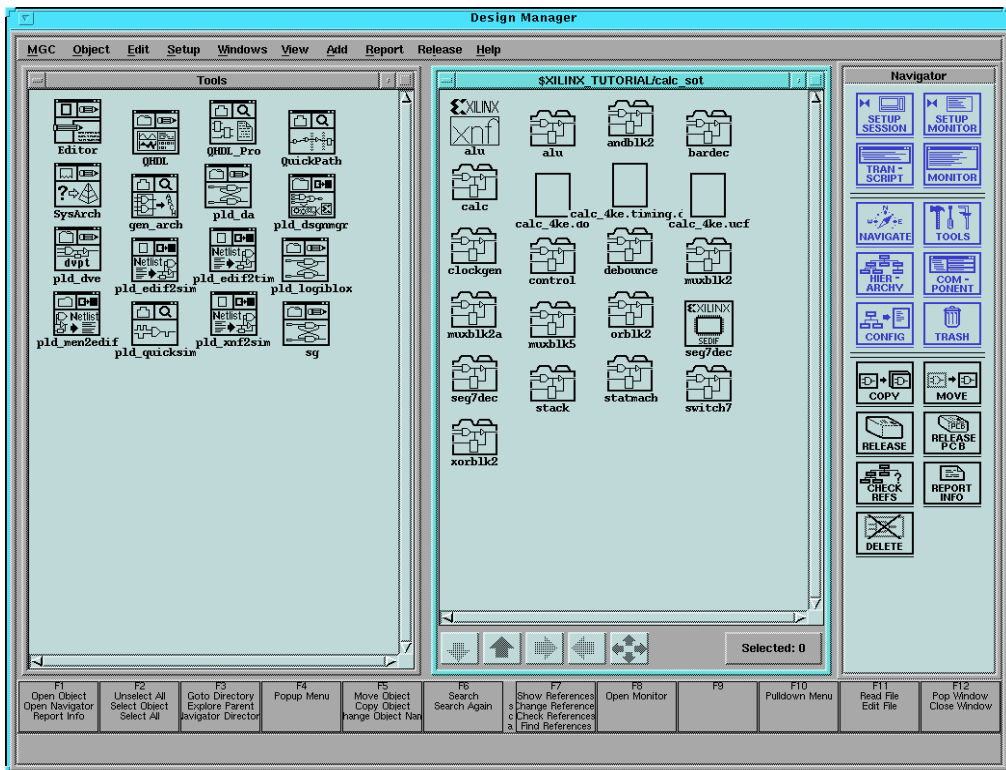


Figure 10-1 Mentor Design Manager Window

Note: This tutorial assumes you are familiar with basic Design Manager procedures. For more information on these procedures, see the [“Schematic Design Tutorial” chapter](#).

Copying the Tutorial Files

The schematic-on-top Calc tutorial files are located in the `$XILINX/mentor/tutorial/calc_sot` directory. To use the Copy operation in Design Manger to copy these files to a local area, perform the following steps:

1. In the Navigator window, move to the directory where the tutorial files were installed.
2. Select the `calc_sot` directory.
3. To see the references in this design, Choose **Right Mouse Button** → **Report** → **Show References** → **For Design**.

A List of Unique References underneath the `calc_sot` directory is displayed. An example reference item might be:

```
$XILINX_TUTORIAL/calc_sot/alu/alu:mgc_symbol[6]
```

This indicates that an ALU symbol (version 6 under Mentor Graphics' versioning system) is referenced by the path `$XILINX_TUTORIAL/calc_sot/alu/alu`.

All references in the design should contain either `$LCA` or `$XILINX_TUTORIAL`.

4. Close the List of Unique References window.
5. With the `calc_sot` directory selected in the Navigator window, choose **Right Mouse Button** → **Edit** → **Copy**.
6. In the dialog box that appears, type the directory path where you want to copy the working copy of the tutorial files.

For example, to copy the files to `/home/dum/tutor/mentor`, enter the following:

```
/home/dum/tutor/mentor/calc_sot
```

7. To keep the design references intact, select **Options** → **Convert References?** **No** from the Copy dialog box.

Normally, you allow Design Manager to update design references when you copy a design directory. In this case, however, you should leave the `$XILINX_TUTORIAL` reference intact. To make sure the proper directory is referenced, modify the `MGC_LOCATION_MAP` accordingly after you copy the tutorial design.

8. Click **OK** to exit the Options dialog box.
9. Click **OK** again to exit the Copy dialog box and start the Copy process.
10. Use the Navigator to change directories to the location of the working copy of `calc_sot`. (In the example above, you would click the **four-arrow** button at the bottom of the Navigator window, then type `/home/dum/tutor/mentor/calc_sot` in the dialog box.)
11. Modify your `MGC_LOCATION_MAP` file so that the `$XILINX_TUTORIAL` variable points to the directory where the copy of `calc_sot` is located. In the example above, change the `$XILINX_TUTORIAL` section of the file so that it reads:

```
$XILINX_TUTORIAL  
/home/dum/tutor/mentor
```

12. Read the newly modified location map into Design Architect by selecting **MGC → Location Map → Read Map** from the menu bar.
13. In the dialog box that opens, type:

```
$MGC_LOCATION_MAP
```
14. Click **OK**.

The `$XILINX_TUTORIAL` soft name now points to the new tutorial area.

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 `$XILINX_TUTORIAL/calc_sot` in the Directory field of the dialog box, then select **OK** or press return. This sets the working directory to the directory where you work on the tutorial.
3. Select the `$XILINX_TUTORIAL/calc_sot/calc` design object in the Navigator window.

4. Select **Right Mouse Button** → **Open** → **p1d_da**.

The Design Architect window appears and displays the Calc design as shown in the following figure.

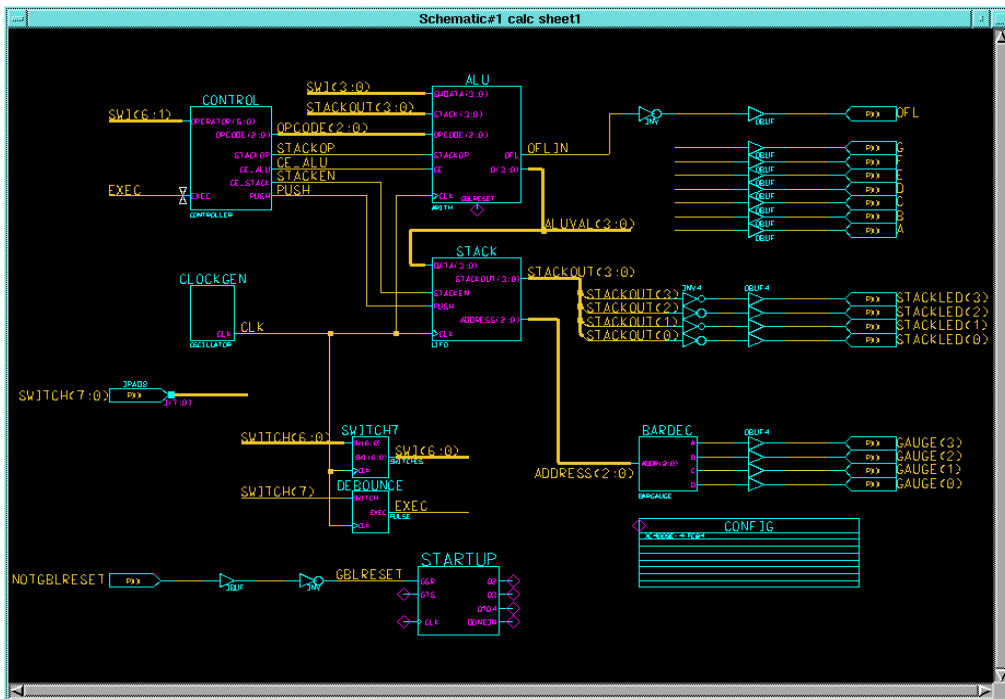


Figure 10-2 Top-Level Schematic for Calc

5. Resize the Design Architect window to cover the entire screen.

Completing the Calc Design

To complete the tutorial design, you need to link VHDL entities to symbols in the schematic using QVHcom and Design Architect.

If you need to stop the tutorial at any time, be sure to save your work as follows:

1. Select **Check** → **Sheet** from the menu bar.

A window appears containing the results of the design rule check.

2. After reviewing the contents of this window, close it and reselect the schematic window.

Warning: It is important to check your design first before saving it.

3. Select **File** → **Save** from the menu bar to save the design.

Design Description

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. A count of the items in the stack is displayed as a “gauge” on another bar LED.

In this tutorial, you create a new symbol for the SEG7DEC component from its associated seg7dec.vhd description, then instantiate that symbol onto the Calc schematic. You then link an existing ALU symbol to its associated alu.vhd description. A CONFIG block and a STARTUP block have already been added to the Calc design as well. For more information on using CONFIG and STARTUP, see the [“Schematic Design Tutorial” chapter](#).

The design consists of the following functional blocks:

- **ALU**—The arithmetic functions of the processor are performed in this block. This block is defined by the VHDL file alu.vhd.
- **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 synchronous RAM in the XC4000E design.
- **DEBOUNCE**—This circuit debounces the “execute” switch, providing a one-shot output.
- **SEG7DEC**—This block decodes the output of the ALU for display on the 7-segment decoder. You generate the symbol for this module from its behavioral VHDL description in seg7dec.vhd.
- **CLOCKGEN**—This block uses an internal oscillator circuit in XC4000E devices to generate the clock signal.

- **BARDEC**—This block shows how many items are on the stack on a “gauge” of four LEDs.
- **SWITCH7**—This is a user-defined module consisting of seven input flip-flops used to latch the switch data.

Note: Basic Xilinx design concepts such as assignment of pin properties, use of STARTUP and CONFIG, and incremental design methodology as well as details about Xilinx device architecture are not covered in this chapter. For more information on these topics, see the “[Schematic Design Tutorial](#)” chapter.

Adding the SEG7DEC Component

On the Calc schematic, notice a space near the upper-right corner of the schematic between the ALUVAL(3:0) bus and the inputs to seven OBUF elements that feed LED outputs A through G. This is where the SEG7DEC symbol must be placed.

This component has a VHDL file, `seg7dec.vhd`, that describes its behavior. In this exercise, you compile the `seg7dec.vhd` file for simulation, generate a symbol for it, then instantiate the entity into the Calc schematic so that it may be simulated and built into the implemented device.

Compiling the VHDL Entity

To compile the VHDL file for SEG7DEC, follow these steps:

1. Take a look at the `src/seg7dec.vhd` file with the text editor you normally use.

This entity includes a case statement that decodes a four-bit number into a set of seven signals suitable for display on a seven-segment LED display. The `src/seg7dec.vhd` file is as follows:

```
process (Q)
begin
  case Q is
    when "0000" => DISPLAY <= "0000001";
    when "0001" => DISPLAY <= "1001111";
    when "0010" => DISPLAY <= "0010010";
    when "0011" => DISPLAY <= "0000110";
    when "0100" => DISPLAY <= "1001100";
    when "0101" => DISPLAY <= "0100100";
    when "0110" => DISPLAY <= "0100000";
```



```
when "0111" => DISPLAY <= "0001101";
when "1000" => DISPLAY <= "0000000";
when "1001" => DISPLAY <= "0000100";
when "1010" => DISPLAY <= "0001000";
when "1011" => DISPLAY <= "1100000";
when "1100" => DISPLAY <= "0110001";
when "1101" => DISPLAY <= "1000010";
when "1110" => DISPLAY <= "0110000";
when others => DISPLAY <= "0111000";
end case;
end process;
```

2. Close the src/seg7dec.vhd file.
3. To create a VHDL work library where compiled entities will reside, type the following at the system prompt from within the \$XILINX_TUTORIAL/calc_sot directory:

```
qhlib work
```

A library directory called “work” now exists in the tutorial project directory.

4. Map this directory to “work” so that the VHDL compilation programs can recognize it:

```
qhmap work work
```

You should now have a new file in your tutorial directory called quickhdl.ini, which contains the following library entry:

```
[Library]
work = work
```

This allows the VHDL-simulation programs to recognize this as the working directory.

5. Compile the src/seg7dec.vhd file with the QVHCOM command:

```
qvhcom -qhpro_syminfo src/seg7dec.vhd
```

The -qhpro_syminfo option tells QVHCOM to write out information needed by the Design Architect Symbol Generator.

6. With the Calc schematic still open in Design Architect, select **Miscellaneous** → **Generate Symbol** from the menu bar.
7. In the Generate Symbol dialog box that appears, select **Choose Source** → **Entity**.

The dialog box fields change as shown in the Generate Symbol dialog box.

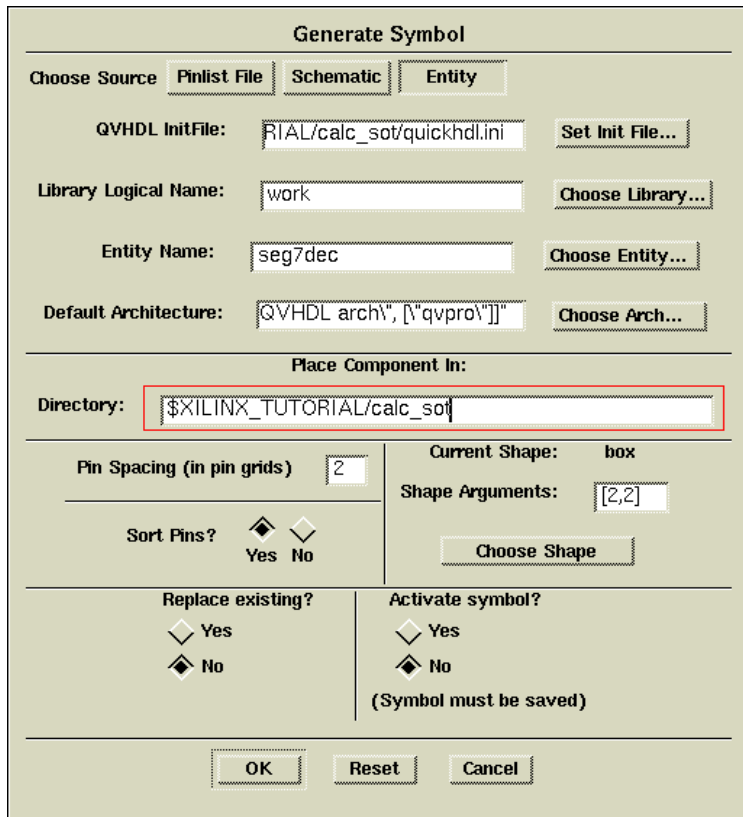


Figure 10-3 Generate Symbol Dialog Box

8. In the Generate Symbol dialog box, make sure the fields are set as shown in the following table:

Table 10-2 Generate Symbol Settings for SEG7DEC

Field	Value
QVHDL InitFile	\$XILINX_TUTORIAL/calc_sot/quickhdl.ini
Lib. Logical Name	work
Entity Name	seg7dec
Deft. Architecture	["behavior", "QVHDL arch", ["qvpro"]]

Table 10-2 Generate Symbol Settings for SEG7DEC

Field	Value
Place Comp. in Dir.	\$XILINX_TUTORIAL/calc_sot
Pin Spacing	2
Shape Arguments	[2,2]
Sort Pins?	Yes
Replace existing?	No
Activate symbol?	No

Note: You can select the architecture setting from a list by click on the **Choose Arch** button. Since only one architecture has been compiled for the SEG7DEC entity, you may also leave this field blank.

9. Click **OK**.

After a few moments, the Symbol Editor appears with the newly created SEG7DEC component. Note the properties attached to this symbol and its pins. These properties allow the underlying entity and the upper-level schematic portion to be simulated concurrently in QuickHDL Pro.

Note: If you get the error “Entity source work_library/_parsed.vhd does not exist,” make sure you specified the -qhpro_syminfo option on the QVHcom command line.

10. Add “SEG7DEC” text to the symbol as shown. (If you do not know the procedure for this, refer to the instructions in the **“Schematic Design Tutorial” chapter**.)

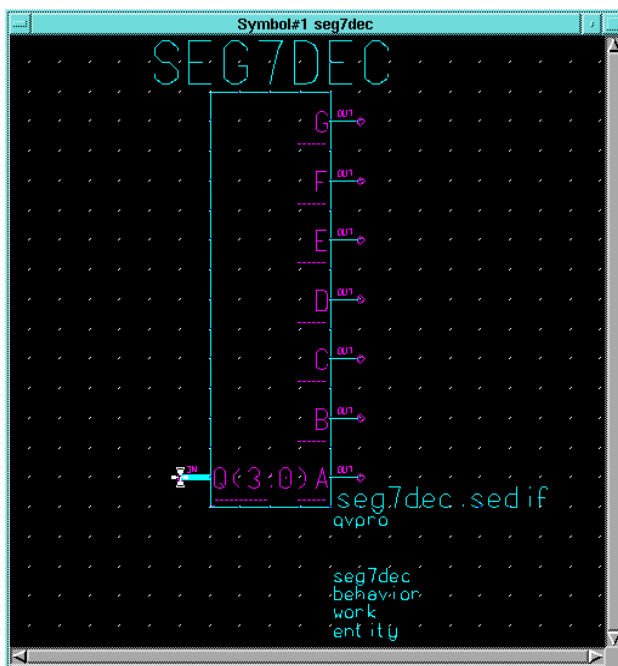


Figure 10-4 Generated SEG7DEC Symbol

All symbols that have an associated non-schematic model must have a FILE property attached to them so that the Xilinx netlister (NGDBUILD) can incorporate these portions of the design into the implemented design. The value of this property is the actual filename of the submodule netlist. This filename can have one of several different extensions, based on the netlist format it uses.

Table 10-3 Common FILE Property Extensions

Extension	Netlist Format
.edif	Generic Xilinx-compatible EDIF 2.0
.xnf	Generic XNF (Xilinx Netlist Format) 6.x
.sedif	Synopsys Xilinx-compatible EDIF 2.0
.sxnf	Synopsys XNF 6.x

The presynthesized SEG7DEC module included with this tutorial was generated by Synopsys' Design Compiler, which uses SEDIF.

- Select the body of the SEG7DEC symbol and add the following property:

Name: **FILE**

Value: **seg7dec.sedif**

- Check and Save the symbol.

Now that a symbol exists for this component, you can instantiate it onto the top-level Calc schematic.

- With the Calc schematic window active, click **CHOOSE SYMBOL** from the Schematic Palette and select the **seg7dec** component.
- Instantiate this component between the ALUVAL(3:0) bus and the nets that lead to the outputs A-F as shown below.

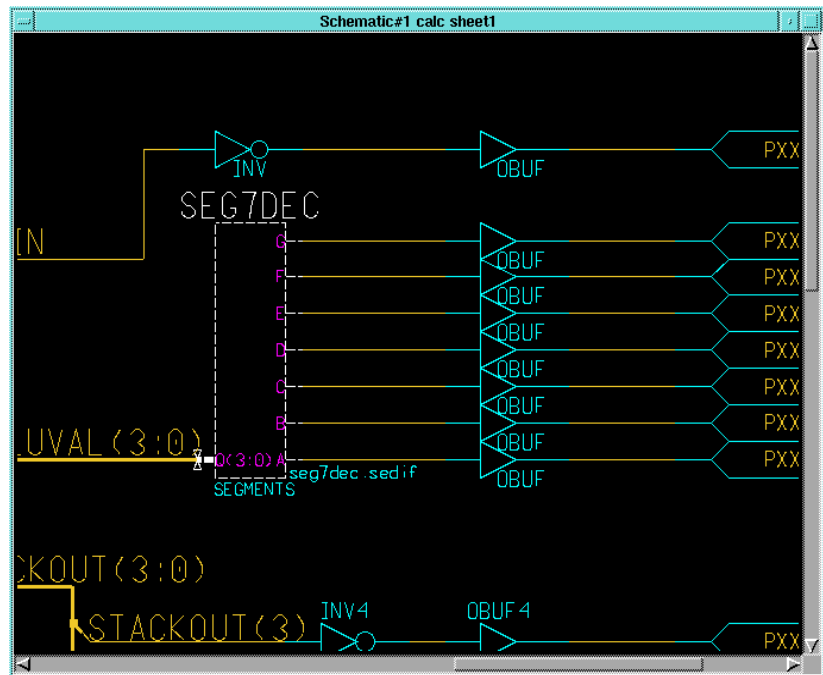


Figure 10-5 Adding the SEG7DEC Symbol

- Add the instance name **SEGMENTS** to the SEG7DEC symbol. (Add property **INST**, value **SEGMENTS**.)
- Check and Save the Calc schematic. Leave the schematic open.

Linking a VHDL Entity to the ALU Component

VHDL can also be associated with a pre-existing symbol. The following procedure links the `alu.vhd` entity with the ALU symbol, which has already been instantiated on the Calc schematic.

As with `SEG7DEC`, ALU has a VHDL file, `alu.vhd`, that describes its behavior. In this exercise, you compile the `alu.vhd` file for simulation, link the compiled model to the existing ALU symbol, then update the instantiated entity in the Calc schematic so that it may be simulated and built into the implemented device.

Compiling the VHDL Entity

To compile the VHDL file for ALU, follow these steps.

1. Take a look at the `alu` or `seg7dec.vhd` file with the text editor you normally use.

This entity includes a four-bit data register, several multi-bit gate functions (AND, OR, and XOR), and an adder-subtractor. To model the device's system-wide global set/reset, an additional port, `GBLRESET`, has been added from the schematic-based ALU design. (See the [“Schematic Design Tutorial” chapter](#).) This signal is brought into the process block that controls the four-bit data register:

VHDL ALU Register Description:

```
process (CLK, GBLRESET)
begin
  if (GBLRESET='1') then
    QIN <= "0000";
    OFL <= '0';
  elsif (CLK'event and CLK='1') then
    if (CE='1') then
      if (QRESET='1') then
        QIN <= "0000";
        OFL <= '0';
      else
        QIN <= MUX;
        OFL <= OVER;
      end if;
    end if;
  end if;
end process;
```

The GBLRESET signal is written as an explicit asynchronous clear. This allows you to connect the registers in the ALU entity to the system-wide global-set/reset signal from the schematic. In an all-schematic design, the system-wide global-set/reset net does not need to be connected, since it is implicitly connected to all flip-flops in a device, both for simulation and implementation. For simulation purposes, however, this signal needs to be explicitly connected to all flip-flops in all VHDL modules. The Xilinx Core Tools recognize this as a redundant connection and subsequently trim it out of the implemented design.

2. When you are finished perusing the ALU VHDL description, close the file.

Since a work directory already exists for compiling VHDL modules, you can go directly to compiling the ALU entity.

3. Compile the `src/alu.vhd` file with the QVHCOM command:

```
qvhcom -explicit -qhpro_syminfo src/alu.vhd
```

The “-explicit” option allows QVHCOM to tolerate multiply-defined standard functions. This is required because the equality operator (“=”) is defined in both the `ieee.std_logic_1164` and `ieee.std_logic_unsigned` packages, both of which are called out in the `alu.vhd` file.

4. With the Calc schematic still open in Design Architect, select the ALU symbol.
5. Open down into the symbol by selecting **Right Mouse Button** → **Open Down**.
6. Select **symbol:alu**.
7. Click **OK**.

The Symbol Editor appears.

If you performed the Schematic Design Tutorial with the all-schematic-based Calc design, you should notice that the ALU symbol has an extra pin, GBLRESET, that corresponds to the extra VHDL port mentioned above.

8. From the menu bar, select **File** → **Import VHDL Entity**.

The Import VHDL Entity dialog box appears.

Figure 10-6 Import VHDL Entity Dialog Box

9. Set the following values in the dialog box:

Table 10-4 Import VHDL Entity Settings for ALU

Field	Value
QVHDL InitFile	\$XILINX_TUTORIAL/calc_sot/quickhdl.ini
Lib. Logical Name	work
Entity Name	seg7dec
Deflt. Architecture	["behavior", "QVHDL arch", ["qvpro"]]

Note: You can select the architecture setting from a list by clicking on the **Choose Arch** button. Since only one architecture has been compiled for the ALU entity, you may also leave this field blank.

10. Click **OK**.

In the Symbol Editor, the ALU symbol's body and pins are annotated with properties similar to those that Generate Symbol attached to the SEG7DEC symbol. These properties allow the underlying entity and the upper-level schematic portion to be simulated concurrently in QuickHDL Pro.

Note: If you get the error "Entity source *work_library/_parsed.vhd* does not exist," make sure you specified the `-qhpro_syminfo` option on the QVHCOM command line.

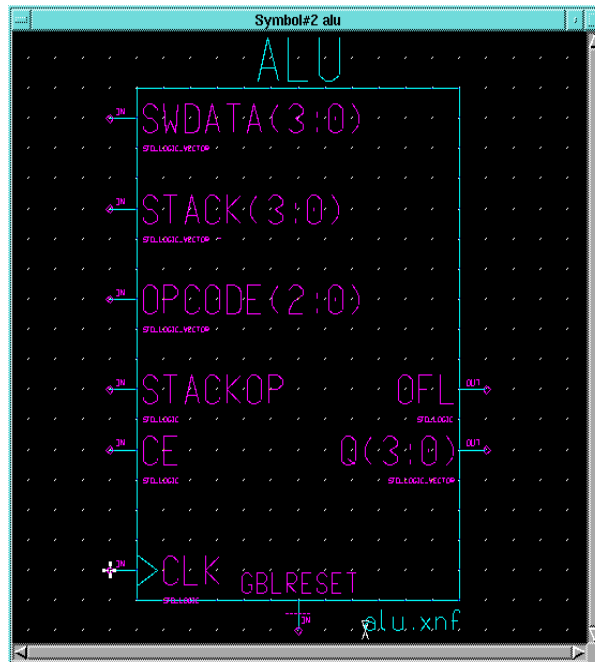


Figure 10-7 ALU Symbol with VHDL-Import Properties

Since the ALU module is also a non-schematic element, you must attach a FILE property to it so the Xilinx netlister (NGDBUILD) can incorporate this portion of the design into the implemented netlist. The ALU netlist included with this tutorial was synthesized by Exemplar's Galileo Logic Explorer, and thus uses generic XNF.

11. Select the body of the ALU symbol and add the following property:

Name: **FILE**
Value: **alu.xnf**

12. Check and Save the symbol.

Since the symbol has now changed, you must reflect the change in the top-level Calc schematic.

13. With the Calc schematic window active, select the ALU symbol, then select **Right Mouse Button** → **Update** → **Auto**.

This updates the ALU instantiation with the new symbol properties.

The GBLRESET pin on the ALU symbol must be attached to the global set/reset signal from the top-level design. This signal is the one connected to the GSR input on the STARTUP block. In the case of Calc, this net is also called GBLRESET.

14. Attach a new net to the GBLRESET pin and name it GBLRESET.

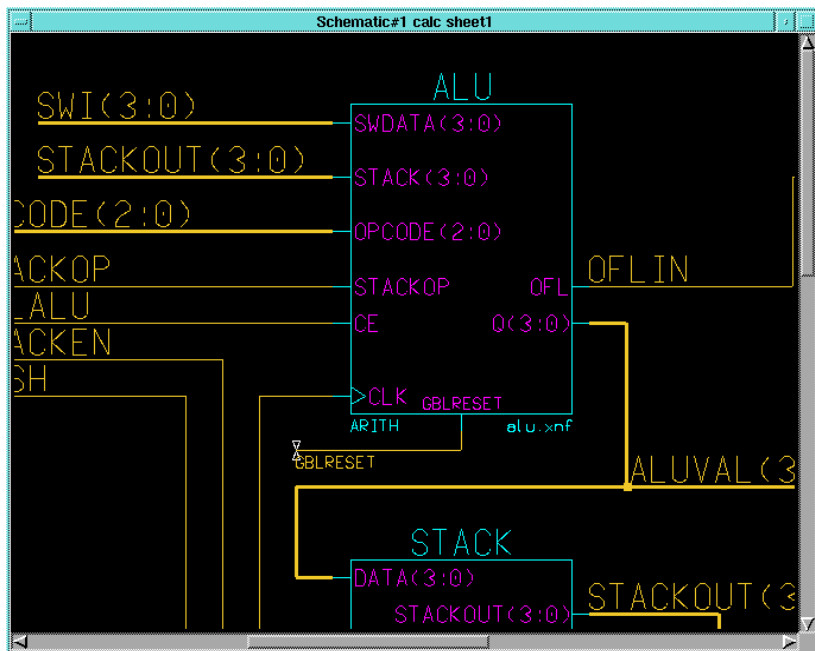


Figure 10-8 Updating the ALU Symbol

15. Check and Save the Calc schematic.
16. Exit Design Architect.

Using a Constraints File

Using a constraints file, you can supply constraints information in a textual form. An example of a constraints file is shown below. The example shows the user constraints file, calc_4ke.ucf, that is supplied

with this tutorial. The constraints file syntax is the same for all device families.

Note: You may also place location constraints directly on the schematic. For more information, see the [“Schematic Design Tutorial” chapter](#).

The place and route software must be instructed to read and apply the .ucf file when the design is read into the Xilinx Design Manager. The procedure for doing this is detailed later in the [“Using the Xilinx Design Manager” section](#).

Example Constraints File:

```
# CALC_4KE.UCF
# User constraints file for CALC, XC4003E-PC84
# Uses angle brackets, as per PLD_MEN2EDIF option

NET SWITCH<7>      LOC=P19;
NET SWITCH<6>      LOC=P20;
NET SWITCH<5>      LOC=P23;
NET SWITCH<4>      LOC=P24;
NET SWITCH<3>      LOC=P25;
NET SWITCH<2>      LOC=P26;
NET SWITCH<1>      LOC=P27;
NET SWITCH<0>      LOC=P28;

NET A              LOC=P49;
NET B              LOC=P48;
NET C              LOC=P47;
NET D              LOC=P46;
NET E              LOC=P45;
NET F              LOC=P50;
NET G              LOC=P51;
NET OFL           LOC=P41;

NET GAUGE<3>       LOC=P61;
NET GAUGE<2>       LOC=P62;
NET GAUGE<1>       LOC=P65;
NET GAUGE<0>       LOC=P66;

NET STACKLED<3>    LOC=P57;
NET STACKLED<2>    LOC=P58;
NET STACKLED<1>    LOC=P59;
NET STACKLED<0>    LOC=P60;
```

NET NOTGBLRESET LOC=P567

Performing Functional Simulation

Functional simulation is performed before design implementation to verify that the schematic that you have designed is logically correct. All components in the Calc design have built-in simulation models so little pre-processing is necessary. However, every top-level schematic design in Mentor Graphics must have a simulation viewpoint before you can simulate it in QuickHDL Pro. The viewpoint describes how a design should be interpreted, including what components in the design are primitives, as well as how components within the design hierarchy should be modeled.

Note: The following instructions refer to two related but different programs: QuickHDL and QuickHDL Pro. QuickHDL is Mentor Graphics' HDL-only simulator, while QuickHDL Pro is a co-simulation tool that runs both QuickHDL to simulate the HDL portions of a design and QuickSim to simulate the schematic portions. QuickHDL Pro runs a process known as a FlexSim backplane through which the two “solvers” (QuickHDL and QuickSim) communicate with each other.

Also note that the instructions that follow do not have detailed information on basic QuickSim operations such as selecting nets and displaying Trace and List windows. For information on these procedures, please refer to the [“Schematic Design Tutorial” chapter](#).

Using Pld_dve

To use the PLD Design Viewpoint Editor to generate a design viewpoint to tell QuickSim (as run by QuickHDL Pro) how to interpret certain Xilinx-specific design properties, follow these steps.

1. Select the calc design object from the appropriate directory in the Navigator window.
2. Invoke `pld_dve` on the design by selecting **Right Mouse Button** → **Open** → `pld_dve`.

A dialog box appears. Note that the component name, Calc, is entered automatically with a fully qualified path.

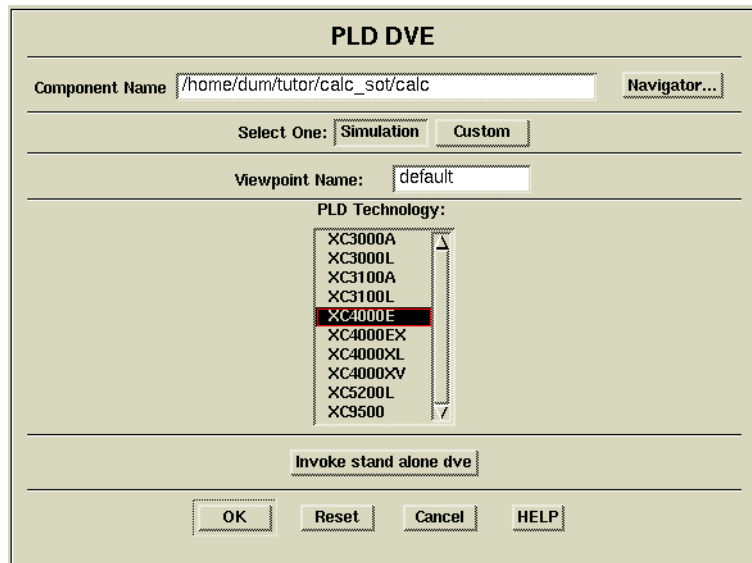


Figure 10-9 Invoking Pld_dve for Functional Simulation

3. Select the XC4000E PLD Technology from the listing as shown in the figure above. (Leave other options set to their defaults, as shown in the figure.)
4. Click OK.
The pld_dve script executes.
5. Once pld_dve completes, dismiss the shell window in which it executed.

Invoking QuickHDL Pro

To invoke QuickHDL Pro for functional simulation on the Calc design, follow these steps:

1. Select the Calc design object in the Navigator window.
2. Invoke QuickHDL Pro on the design by selecting **Right Mouse Button** → **Open** → **QHDL_Pro**.

A dialog box appears. Note that the component name, Calc, is entered automatically with a fully qualified path.

3. Since the top-level Calc design is a schematic (EDDM model), verify that EDDM Design is the design set to Invoke on.

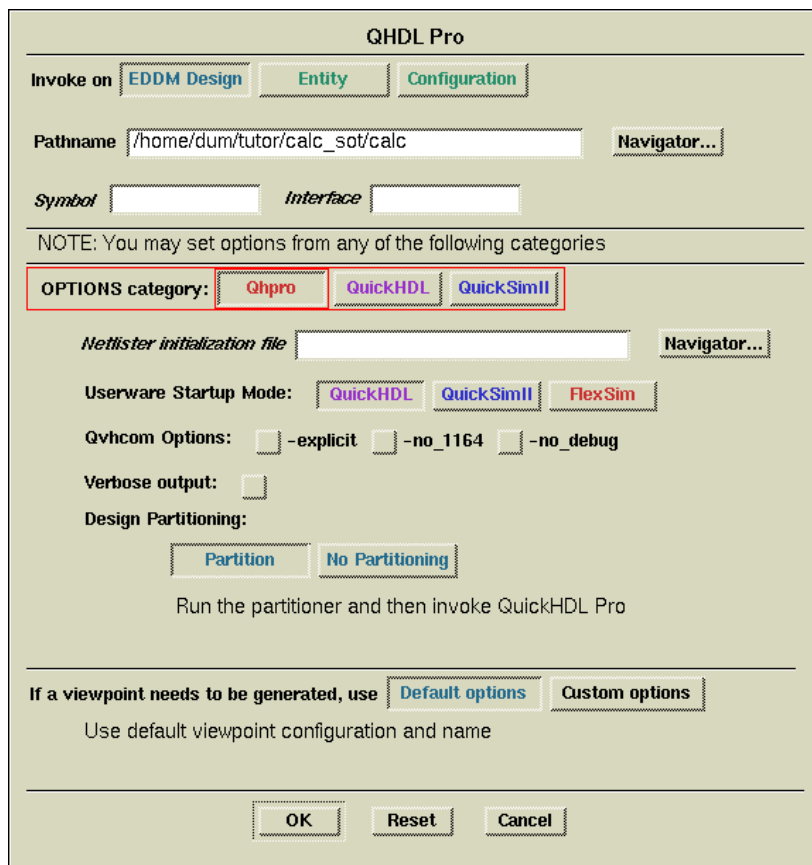


Figure 10-10 Invoking QuickHDL Pro for Functional Simulation

Note the **OPTIONS** category buttons. These buttons allow you to set options for QuickHDL Pro, as well as for each of the two simulation “solvers” (QuickHDL and QuickSim). Each button brings up its own set of options in the **OPTIONS** panel. Each set of options is independent of the other two sets of options, and options in all three panels are applied concurrently when QuickHDL Pro is run.

4. In the QuickHDL Pro dialog box, select **OPTIONS category** → **QuickHDL** to see the QuickHDL options.

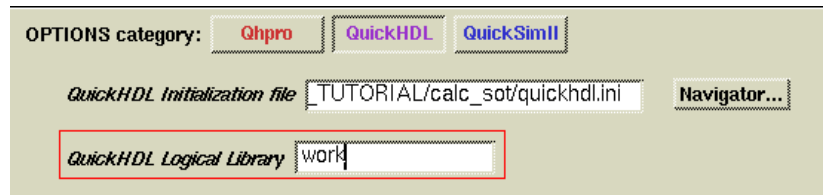


Figure 10-11 QuickHDL Options

5. Under QuickHDL Initialization file, enter:
`$XILINX_TUTORIAL/calc_sot/quickhdl.ini`
6. Under QuickHDL Logical Library, enter:
`work`

Note: The QuickHDL initialization file defaults to the quickhdl.ini file in the working directory. The QuickHDL logical library defaults to the “work” directory as called out in the initialization file.

7. In the QuickHDL Pro dialog box, select **OPTIONS category** → **QuickSim** to see the QuickSim options.

The Unit timing mode should already be set, since this is the default. This tells the QuickSim solver to run in functional-simulation mode.

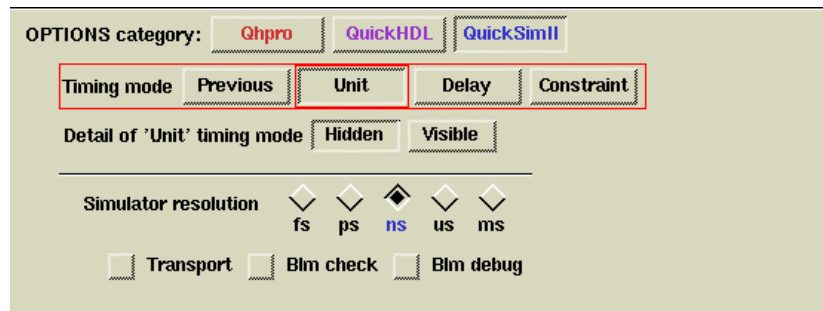


Figure 10-12 QuickSim Options

8. In the QuickHDL Pro dialog box, select **OPTIONS category** → **QuickHDL** and note that the settings you put in the QuickHDL Options panel are still in place.
9. Click **OK** to invoke the QuickHDL Pro simulator.


```

1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_arith.all;
4  use IEEE.std_logic_unsigned.all;
5
6  -- ALU: Arithmetic Logic Unit (Calc Tutorial)
7  -- Written by Xilinx Applications, 10 June 1996
8  -- Copyright 1996 Xilinx
9
10 entity alu is
11     port (SMDATA:      in STD_LOGIC_VECTOR (3 downto 0);
12          STACK:       in STD_LOGIC_VECTOR (3 downto 0);
13          OPCODE:      in STD_LOGIC_VECTOR (2 downto 0);
14          STACKOP:     in STD_LOGIC;
15          CE, CLK:     in STD_LOGIC;
16          GBLRESET:   in STD_LOGIC;
17          OFL:        out STD_LOGIC;
18          Q:          out STD_LOGIC_VECTOR (3 downto 0));
19 end alu;

```

Figure 10-14 ALU Source Window

Viewing and Navigating the VHDL Hierarchy

QuickHDL allows you to view the design hierarchy of the HDL portion of a mixed design. To see this hierarchy and its associated signals follow these steps:

1. From the QHPro(QuickHDL) menu bar, select **View** → **Structure**.

The Structure window appears as shown. Since the ALU source code is displayed in the Source window, the ALU entity is highlighted.

2. Select **View** → **Signals** to display the Signals window which shows a list of all signals underneath the ALU entity.

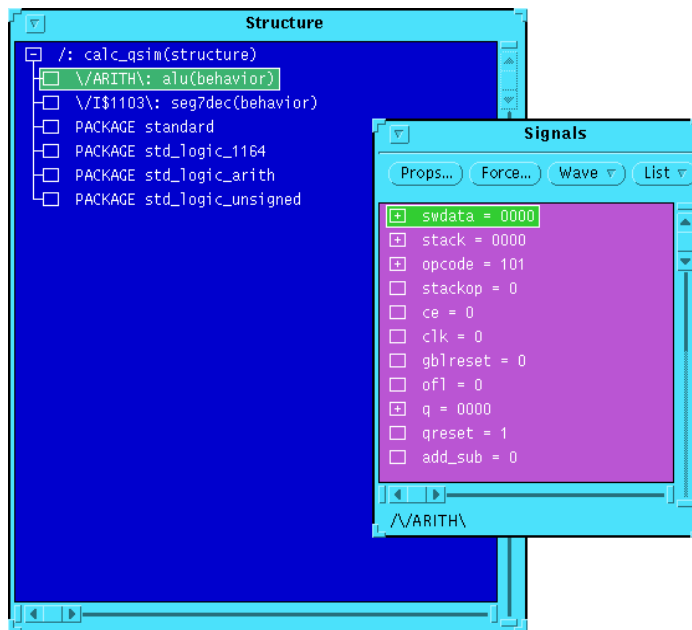


Figure 10-15 Structure and Signals Windows

3. With the left mouse button, select the `seg7dec(behavior)` entity in the Structure window.

Notice that the Source window now changes to display the `SEG7DEC` VHDL file, and the Signals window changes to display the associated signals.

4. Select the top-level `calc_qsim(structure)` entity in the Structure window.

Notice that the Source window displays VHDL code even though the top level was originally drawn on a schematic. This VHDL file was written by QuickHDL Pro upon invocation so that QuickHDL could simulate all instantiated VHDL modules from the top level.

5. Reselect the `alu (behavior)` module in the Structure window.

The Signals listing displays the ALU signals. Note that buses in the Signals listing have “+” icons next to them. If you click on one of these icons, it changes to a “-” icon, and the list expands to include all bit signals within the corresponding bus.

6. Click the “-” icon to collapse the listing.
7. Select the following signals in the Signals window. You may have to scroll down the list to see them all.

```

clk
ce
opcode
q
stack
stackop
swdata

```

You can select a collection of signals by clicking on the first signal name with the left mouse button, then clicking on subsequent signal names with the middle mouse button. If you select a signal name you don't want selected, click on the signal name again with the middle mouse button to unhighlight it.

The waveforms for these signals can be traced in a QuickHDL Wave window.

8. In the Signals window, select **Wave** → **Selected signals**.

The Wave window appears as shown. Since the simulation has not yet been run, no waveforms are displayed.

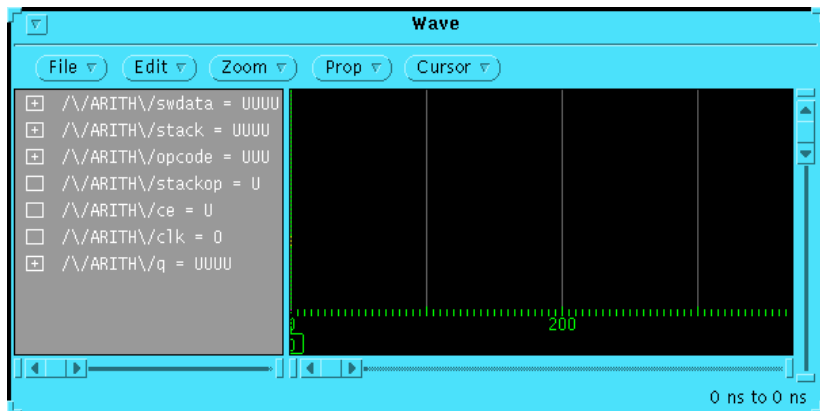


Figure 10-16 Wave Window at Time 0

In the next section, you complete the functional simulation and view the waveforms in the Trace (QuickSim) and Wave (QuickHDL) windows.

Completing the Functional Simulation

You can automate the simulation by using a QuickSim command file. A “dofile,” `calc_4ke.do`, has been supplied with this tutorial for this purpose. This file opens Trace and List windows, sets up simulation vectors, and runs the simulation for a time duration of 3,400 ns. Use it as follows:

1. From the QuickSim menu bar, select **MGC** → **Transcript** → **Replay**.
2. In the dialog box, enter `calc_4ke.do` and click **OK**.

The functional simulation runs automatically with Trace and List windows set up in the QuickSim window. Note that the Wave window under QuickHDL is also updated to reflect the signals underneath the ALU component.

Note: For more specific information on using QuickSim features, see the “**Schematic Design Tutorial**” chapter. The simulation command file executed here is similar to the one run in that chapter.

3. If the signal names and values in the Wave window are not fully visible, drag the divider between the signal list and the waveforms to see more of the signal names.
4. You can use the scroll bars to view different parts of the simulation in the Wave window. You can also zoom in and zoom out by selecting from the Zoom menu.
5. As with the Signals windows, the signal listing in the Wave window is expandable and collapsible. Click the “+” icon beside the `\/\ARITH\/swdata` signal name to view the individual signal waveforms in the Wave window.

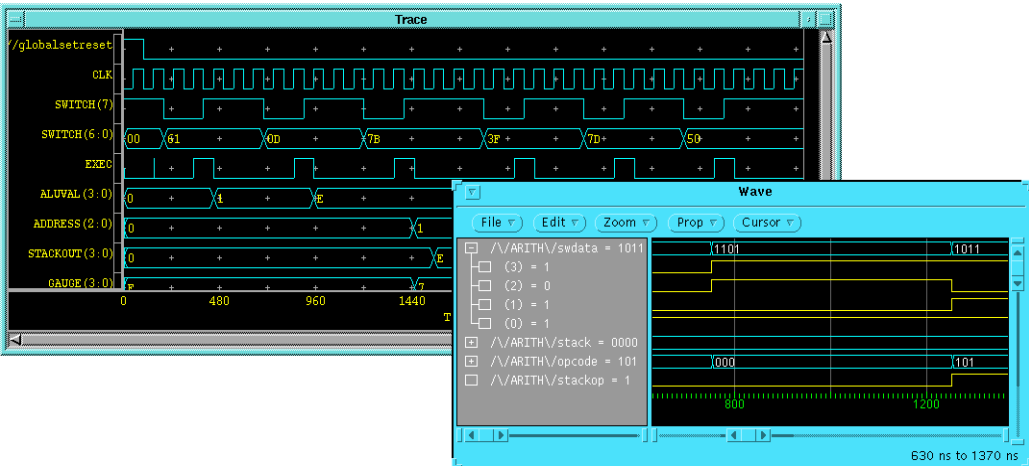


Figure 10-17 QuickSim Trace and QuickHDL Wave Windows

- The SWDATA waveform corresponds to the lower four bits of the SWITCH(6:0) bus on the top-level schematic. SWITCH(3:0) is connected to the SWDATA(3:0) port on the ALU in the top-level schematic.

Verify that the SWDATA value in the Wave window corresponds to the lower four bits of SWITCH(6:0) in the QuickSim Trace window.

Note: You can make this task a easier by selecting the **SWDATA signal** in the Wave window, then choosing **Prop** → **Signal** from the menu bar. Select **Radix: Hex**, then **Apply**. The hexadecimal value displayed in the Wave window now matches up with the lower-order hexadecimal digit from the SWITCH(6:0) value.

- Close the QuickSim and QuickHDL applications by choosing one of the two application windows and selecting **Quit** or **Exit** (depending on your workstation platform) from its control menu (the drop-down menu on the titlebar). It is not necessary to save any simulation results.

Note: Since both QuickSim and QuickHDL are bound to the same process, closing one application automatically closes the other.

Using Pld_men2edif

Once your design is verified to be functionally correct, you use `pld_men2edif`, a tool in the Mentor Graphics Design Manager, to translate your Mentor design into a Xilinx-ready EDIF netlist. Running `pld_men2edif` is always the first step in implementing a design. Whenever you make changes to your schematic, you must run `pld_men2edif` again so that the Xilinx software can process those changes.

When you run `pld_men2edif` from the Mentor Design Manager, the `pld_men2edif` dialog box appears.

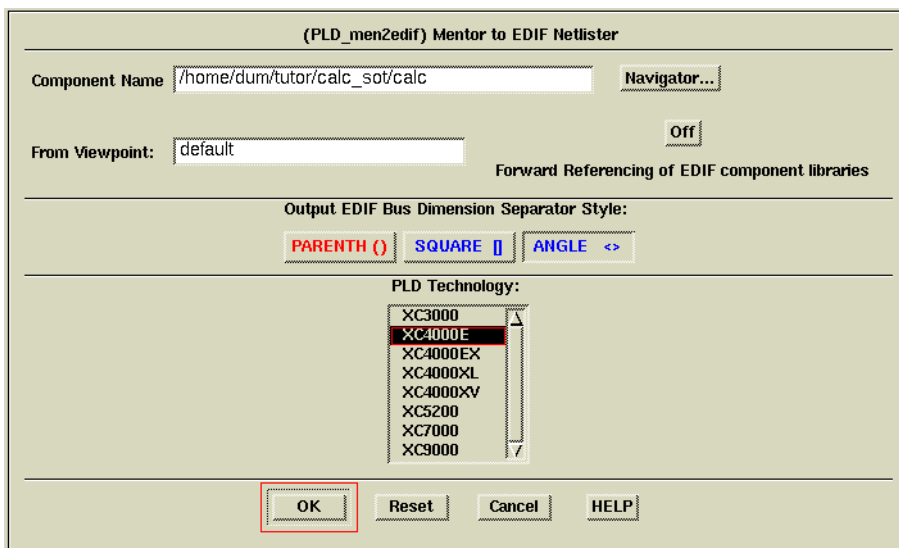


Figure 10-18 Pld_men2edif Dialog Box

Here is an explanation of some of the fields and buttons.

- **Component Name**—Enter the name of the component that you want to process here.
- **From Viewpoint**—If you are an advanced Mentor Graphics designer who uses viewpoints to organize design models and properties, enter the viewpoint name that you wish to use for this EDIF translation. If you do not use or are not familiar with viewpoints, leave this field blank and `pld_men2edif` will use a default value.

- **Forward Referencing of EDIF component libraries**—This option applies only in rare situations where design hierarchy has been structured in such a way that circular or recursive references exist. Normally, this option is set to Off.
- **Output EDIF Bus Dimension Separator Style**—This determines how bus-index delimiters are written into the output EDIF file. This is important if you are merging components from other design-entry tools into a single design. Choosing a bus-index delimiter lets you insure that the bus-index delimiters that `pld_men2edif` writes out are consistent with those of any other design-entry tools with which you are interfacing.

Since this design contains instantiated HDL components from other vendors, make sure that the bus delimiters written for symbols in the schematic are the same as those written in the synthesized HDL netlists; otherwise, NGDBUILD sees a pin mismatch and does not correctly connect buses that pass up or down through levels of hierarchy. Both of the synthesis tools used to generate the `seg7dec.edif` and `alu.xnf` files in this tutorial use angle brackets. If you are using these pre-synthesized files in your tutorial design, change this setting to ANGLE.

- **PLD Technology**—Select the architectural family from this list, in this case XC4000E.
- **HELP**—If the HELP button is clicked, a short help listing is produced by the `pld_men2edif` script.

To create an EDIF netlist for Calc, perform these steps:

1. Double-click on the `pld_men2edif` icon in Design Manager.
2. For the Component Name, type `$XILINX_TUTORIAL/calc_sot/calc` as shown above.
3. Select the **XC4000E** architecture in the PLD Technology field.
4. Select **OK**.

This opens a new shell window where `pld_men2edif` runs and reports its progress. When `pld_men2edif` has completed, the following should appear at the bottom of the shell window:

```
pld_men2edif ended with return code 0
Done.
```

5. Dismiss the `pld_men2edif` shell window by typing **Ctrl-C** in it or by selecting **Close** from the window's control menu (accessed through the button on the left side of the title bar).

Note: The output of `pld_men2edif` may be sent to the window from which the `pld_dmgr` was originally invoked. This behavior is dictated by the `$MGC_TERMINAL_WINDOW` environment variable; see the Mentor Graphics documentation for more details.

Examining Pld_men2edif Output Files

In addition to the EDIF netlist, `pld_men2edif` also creates a `pld_men2edif.log` file. This file contains a transcript of the processing done by `pld_men2edif`. If the program fails to generate an EDIF netlist, any errors encountered are logged in this file.

Examine the `pld_men2edif.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_men2edif`, including an EDIF file for Calc, and a log file for `pld_men2edif`.

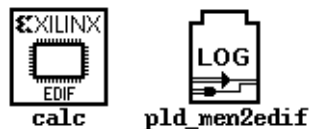


Figure 10-19 Files Created by `Pld_men2edif`

3. Select the LOG icon labeled `pld_men2edif` and choose **Right Mouse Button** → **Open** → **Editor**.

A window appears displaying the log file. When you are done viewing the log, close the window.

Note: You can change the display font in this window by selecting **View** → **Fonts**.

Using the Xilinx Design Manager

The Xilinx Design Manager is a graphical design-flow and project manager. The Xilinx Design Manager takes your design, represented

by the EDIF file from `p1d_men2edif`, and implements it in an FPGA or CPLD. You can also use the Xilinx Design Manager to generate timing information that you can import into QuickSim or QuickHDL.

This section gives a brief overview of the design implementation flow. For a more in-depth discussion of the flow, including advanced implementation options, see the *Development System Reference Guide*.

1. Within the Mentor Design Manager, select the Calc EDIF icon in the Navigator, then select **Right Mouse Button** → **Open** → **p1d_dsgnmgr**.

The Xilinx Design Manager appears as shown. The tool automatically creates a Xilinx project called `calc`. Xilinx project information is kept in a file called `xproject/calc.prj` by default.

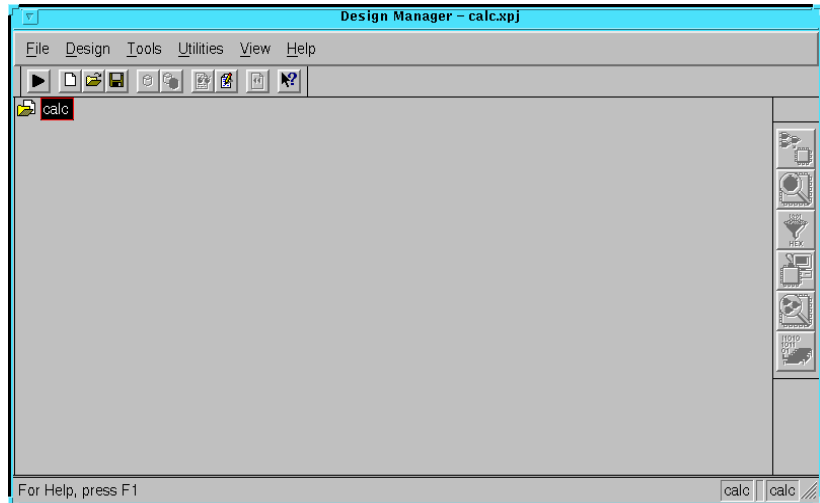


Figure 10-20 Xilinx Design Manager

Each project has associated with it objects known as “versions” and “revisions.” Versions represent logic changes in a design (for example, adding a new block of logic, replacing an AND gate with an OR gate, or adding a flip-flop); revisions represent different executions of the design flow on a single design version, usually with new implementation options (for example, higher place and route effort, a change in part type, or experimentation with new bitstream options). In the next stage, you make a new

version and revision on which you run the implementation design flow.

2. Within the Xilinx Design Manager, select **Design** → **Implement**, which gives you the Implement dialog box, with fields for part type, design version, and revision as shown in the following figure.

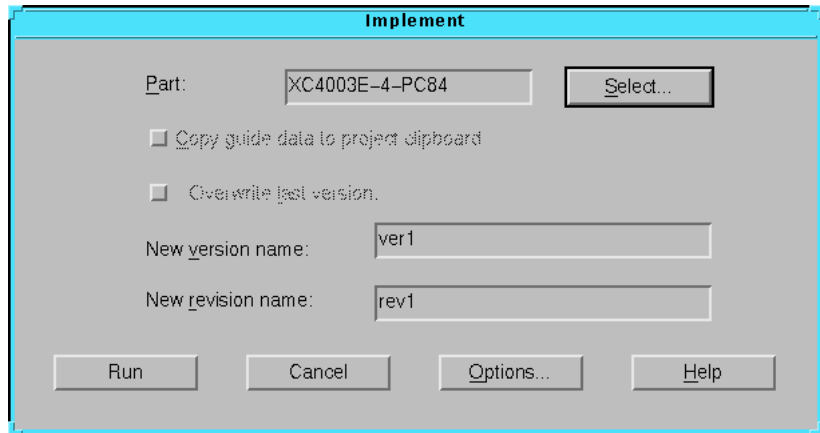


Figure 10-21 Implementation Dialog Box

3. Click the **select** button to display a pull-down listing of available devices.
4. Choose a Family of **XC4000E**, a Device of **XC4003E**, a Package of **PC84**, and a Speed Grade of **-4**.
5. Click **OK**.

The part number is inserted into the Part field in the Implement dialog box.

6. Click on **Options**.

The Options dialog box appears.

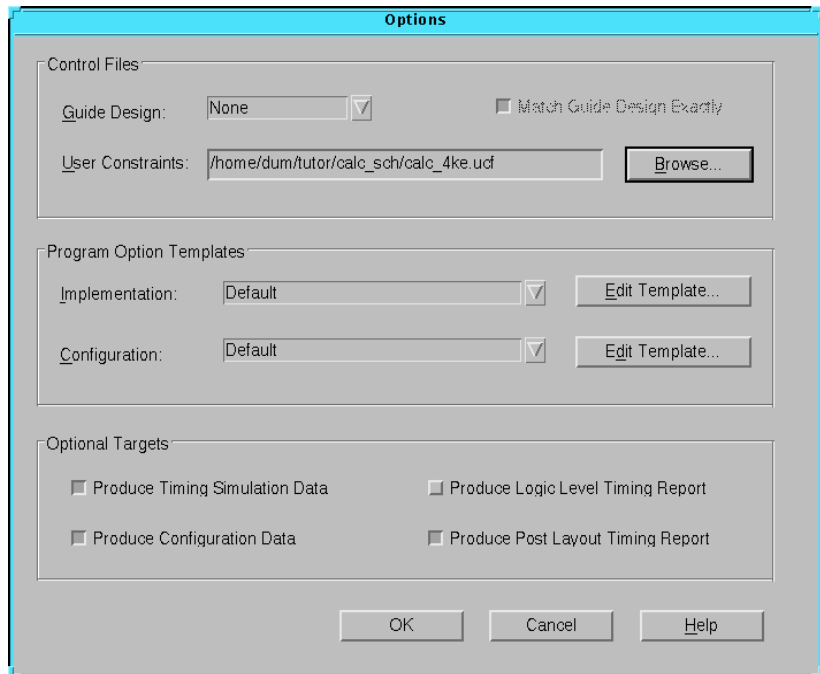


Figure 10-22 Options Dialog Box

7. Click **Browse** by the User Constraints field.
8. Select the `calcul_4ke.ucf` file from the design directory, then Click **OK**.
9. Under Optional Targets, make sure the following are selected:
 - Produce Timing Simulation Data—This generates a back-annotated EDIF netlist that can be imported into the Mentor Graphics tools.
 - Produce Configuration Data—This generates a programming bitstream suitable for downloading into the Xilinx device.
 - Produce Post Layout Timing Report—This generates a timing report file based on how the design is actually routed.

You can also select the following option:

- Produce Logic Level Timing Report—This generates a preliminary (pre-place and route) timing report based on the

number of logic levels in each signal path. Since it is generated before the place-and-route layout step, it does not contain information on device routing. Looking at this report before place and route can be useful for seeing how much “routing slack” you have in a design.

10. Under Program Option Templates Implementation, select **Edit Template**.

The XC4000 Implementation Options dialog box appears as shown in the following figure.

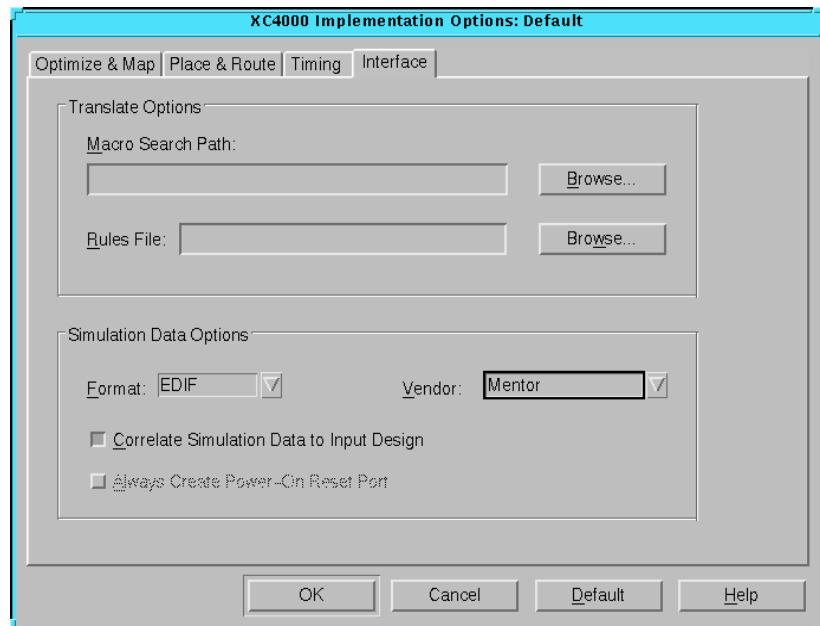


Figure 10-23 Changing EDIF Vendor Information

11. Select the **Interface** tab.
12. In the Interface pane, look under Simulation Data Options and verify that Format is set to **EDIF** and that **Correlate Simulation Data to Input Design** is selected.
13. In the Vendor field, select **Mentor**.
14. Click **OK** to return to the Options window.
15. Click **OK** to return to the Implementation dialog box.

16. Verify that the version is ver1 and the revision is rev1 then click Run.

The Flow Engine comes up as shown in the following figure.

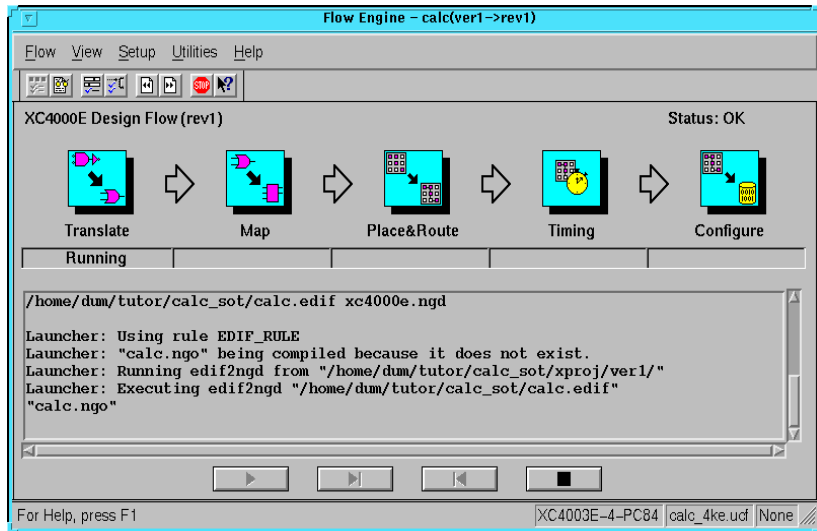


Figure 10-24 The Xilinx Flow Engine

The status bar shows the progress of the implementation flow with the following stages:

- **Translate**—convert the design EDIF file into an NGD (Native Generic Design) file
- **Map**—group basic elements (bels) such as flip-flops and gates into logic blocks (comps); also generate a logic-level timing report if desired
- **Place&Route**—place comps into the device, and route signals between them
- **Timing**—generate timing simulation data and an optional post-layout timing report
- **Configure**—generate a bitstream suitable for downloading into and configuring a device

When the implementation completes, an Implementation Status box appears with:

Implementing revision ver1->rev1 completed successfully.

17. Click on **View Logfile** to display the logfile from Flow Engine.

The report is displayed in vi.

Note: To use another text editor, such as Emacs, as the report viewer, select **File** → **Preferences** from the Xilinx Design Manager.

18. To exit the viewer, type **:q!** and press Return.
19. Click **OK** in the Implementation Status dialog to return to the Xilinx Design Manager.

Performing Timing 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. Also, since the delay-annotated timing netlist is different from the original schematic design, the timing simulation uses a process called *cross-probing* to allow you to view simulation results on your schematic. In this section, you perform a timing simulation of the Calc design by first preparing the design using `pld_edif2tim`. Once this has been done, you run `pld_quicksim` with cross-probing to trace waveforms and annotate results onto your original schematic.

Note: All modules in a schematic-on-top design, including the HDL portions, are written as EDDM models. Since no HDL models are generated for timing simulation, only QuickSim is used to simulate the design, and neither QuickHDL nor QuickHDL Pro is required.

Using Pld_edif2tim to Prepare a Timing Simulation

`Pld_edif2tim` reads a routed EDN file and back-annotates the delays to the schematic. This includes a number of steps, all of which are automatically run by the `pld_edif2tim` script. This script is represented by the `pld_edif2tim` icon in `pld_dmgr`. The files necessary for back-annotation have either been created in the Design Architect tutorial or are included in the solution directories.

Use `pld_edif2tim` to prepare the design for timing simulation as follows:

1. In `pld_dmgr`, use the Navigator to find and select the EDN `time_sim` icon. This represents the timing-annotated netlist generated by the Xilinx Design Manager.

Note: There may be two similar looking types of icons, one marked EDIF and the other marked EDN. An EDIF file represents a netlist translated from the original schematic, while an EDN file represents a netlist translated from a routed NCD file. Be sure you select an EDN file to prepare for timing simulation.

2. Select **Right Mouse Button** → **Open** → `pld_edif2tim`.

A dialog box appears. The Design Manager automatically fills in the dialog box with the name of the EDN file.

3. Select the appropriate PLD technology, for example, XC4000E.
4. Press return or select **OK** to execute the command.

The script produces a shell and runs in it.

Examining the `Pld_edif2tim.log` File

Examine the `pld_edif2tim.log` file as follows:

1. In `pld_dmgr`, select **Right Mouse Button** → **Update Window**.
The window is updated with the files that `pld_edif2tim` generated.
2. Find the `pld_edif2tim` 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 `pld_edif2tim` during the timing flow, see the “**Command Summaries**” section at the end of this chapter. The timing flow is always the same since the starting point is always a routed EDN file with delays.

4. When you have finished looking at the file, close the Editor window.

Using Pld_dve

As with functional simulation, the timing-annotated netlist must also have a viewpoint associated with it.

1. In your design directory, you should now see a directory called `calc_lib`. Double-click on this directory icon to descend into it.
2. Select the `calc` component (which should be at the very bottom on the icon listing) with the left mouse button.
3. Invoke `pld_dve` on the simulation netlist component by selecting **Right Mouse Button** → **Open** → `pld_dve`.

A dialog box appears. Note that the component name, `Calc`, is entered automatically with a fully qualified path.

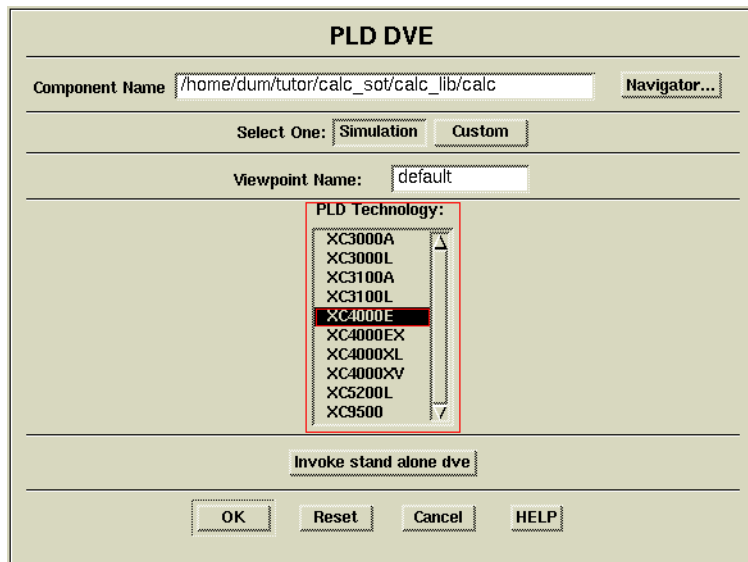


Figure 10-25 Invoking Pld_dve for Timing Simulation

4. Select the appropriate PLD Technology from the listing, *e.g.*, `XC4000E`, as shown in the figure above. (Leave other options set to their defaults, as shown in the figure.)
5. Click **OK** to execute the `pld_dve` script.
6. Once `pld_dve` completes, dismiss the shell window in which it has executed.

Invoking QuickSim for Timing Simulation

1. With the timing-annotated calc component in the calc_lib directory still selected, invoke pld_quicksim by selecting **Right Mouse Button** → **Open** → **pld_quicksim**.

A dialog box appears. Note that the component name, Calc, is entered automatically with a fully qualified path.

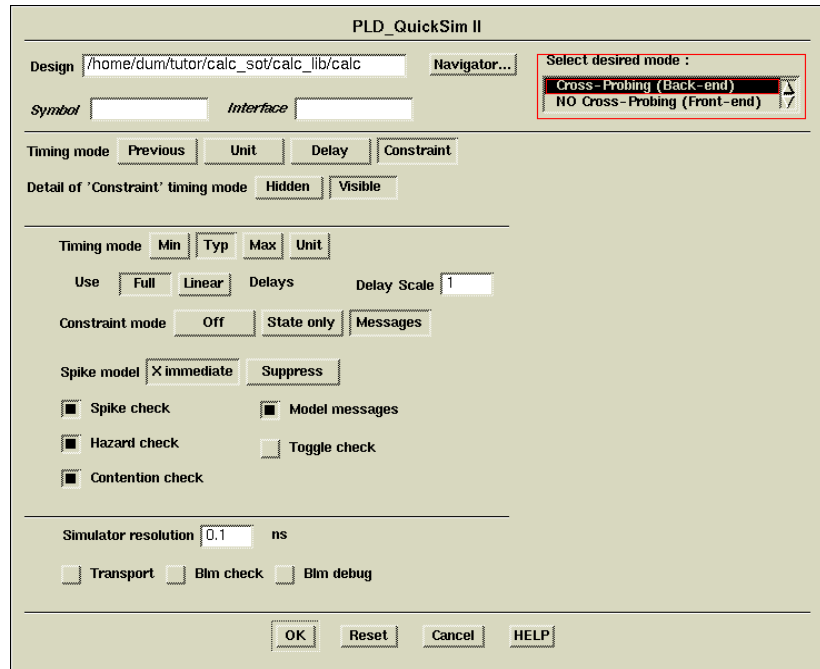


Figure 10-26 Invoking Pld_quicksim for Timing Simulation

2. Select **Cross-Probing** as the desired mode.

This allows QuickSim to use the back-annotated timing model in QuickSim, while allowing you to view the original schematic in DVE. This process is necessary because your original schematic is expressed in Unified libraries, while the back-annotated timing model is generated using simulation primitives.

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.

The Design Viewpoint Editor (DVE) appears and gives a message reading, "To start the cross-probing process, ..."

8. Click **OK** in this message window.
9. Resize the DVE window so that it is almost as large as the entire screen.
10. The QuickSim window also appears. Resize the QuickSim window so that it is almost as large as the entire screen, allowing space for you to click on the DVE window to make it active and display in the foreground.
11. Bring the DVE window to the foreground and select **OPEN DESIGN** from the palette.
12. In the dialog box that appears, enter the name of the *original* component in the Component field, e.g., \$XILINX_TUTORIAL/calc_sot/calc. (Do *not* enter the name of the simulation model, \$XILINX_TUTORIAL/calc_sot/calc_lib/calc.) Click **OK** to load the original viewpoint.

13. Select **OPEN SHEET** from the palette.

The top-level Calc schematic appears.

14. To see the cross-probing process in action, click on the CLK net attached to the CLOCKGEN component.

A few seconds after this net is selected in DVE, a Trace window appears in QuickSim listing the CLK net.

15. Bring the QuickSim window to the foreground to see the Trace window.

16. Select **Transcript** → **Replay** from the QuickSim menu bar.
17. From the dialog box, choose the `calc_4ke.timing.do` file.

This replays a transcript file similar to the one played earlier, except that it does not open the schematic in QuickSim. 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.

The figure below shows how DVE and QuickSim may look like running side by side.

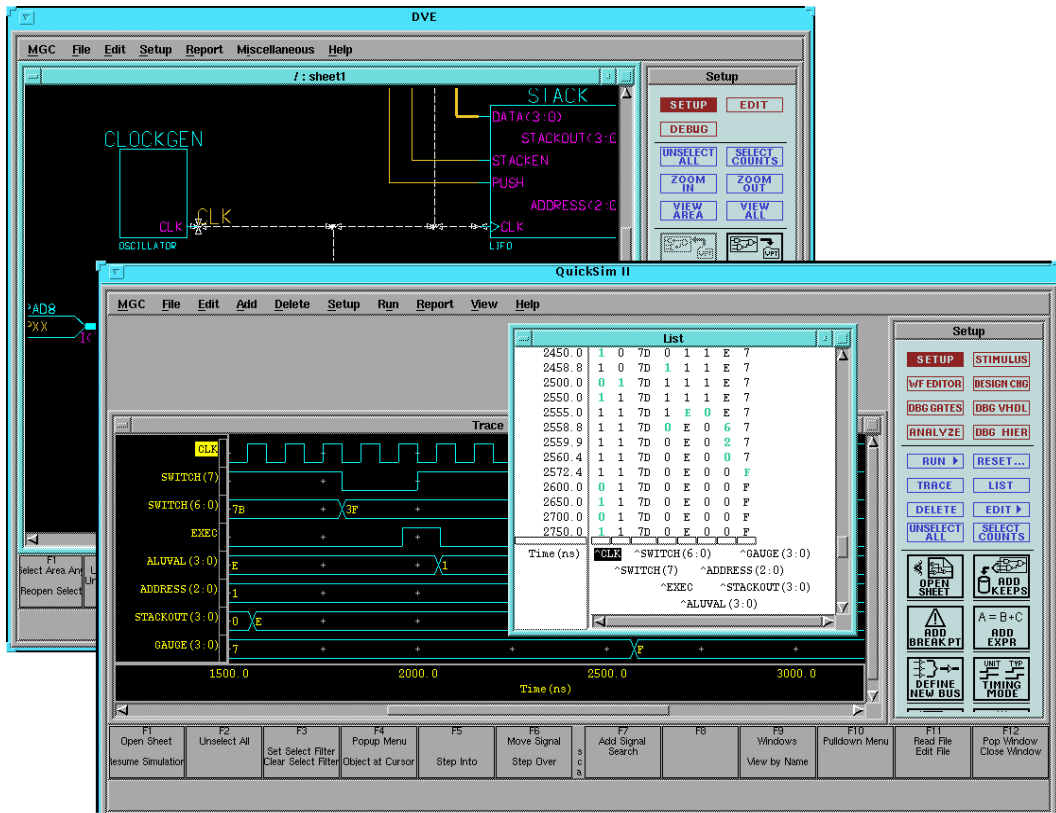


Figure 10-27 Cross-Probing with DVE and QuickSim

18. After examining the waveforms in timing simulation, close both the QuickSim and DVE windows.

Examining Routed Designs with EPIC

At this point in the tutorial, the design process is complete. If you would like to see how the design has been implemented by the Xilinx software, you can take a graphic look at your placed and routed design using the Editor for Programmable Integrated Circuits, or EPIC. You can access EPIC from the Xilinx Design Manager.

EPIC provides several useful functions, such as:

- Manual placement of a pre-routed design
- Manual editing of a routed design
- Static timing analysis



Figure 10-28 EPIC Icon

EPIC is explained in a separate tutorial. See the EPIC Tutorial section of the *EPIC Reference/User Guide*. Before starting this tutorial, be sure to select the **ver1** → **rev1** revision of the design in the project view

Verifying the Design Using a Demonstration Board

Creating and Downloading the Bitstream

A bitstream has been created during the Configure stage in Flow Engine. At this point, you are ready to download the bitstream using a parallel download cable or the more versatile XChecker cable connected to your workstation. The XC4000E version of the Calc design is suitable for download into an FPGA demonstration board available from Xilinx.

Downloading is accomplished with Hardware Debugger. To invoke Hardware Debugger, you select **Tools** → **Hardware Debugger** from the menu bar, or click the Hardware Debugger icon on the toolbar. If you are using an XChecker cable, you can also use the Hardware Debugger to read back information from the device to

verify both the configuration as well as the state of memories and registers within the device.



Figure 10-29 Hardware Debugger Icon

Hardware Debugger is explained in a separate tutorial. See the CALC Tutorial section of the *Hardware Debugger Reference/User Guide*. Before starting this tutorial, be sure to select the `ver1` → `rev1` revision of the design in the project view.

Command Summaries

Although this tutorial uses the Mentor Graphics Design Manager and the Xilinx Design Manager to process the Calc design, you can also manually run the individual programs that these graphical tools run.

This section details command sequences that you can use to perform the translations the Xilinx Design Manager performs in this tutorial. The commands are written as you would type them at the system prompt or in a batch file. You may also see a summary of these system commands by using the **Utilities** → **Command History** and **Utilities** → **Command Preview** selections in Design Manager or Flow Engine. Once you are in the Command History or Command Preview dialog box, select the display Mode to Command Line to see the detailed system commands, including command-line options, used by Flow Engine. You can cut and paste from these Command dialog boxes into your text editor to create batch files.

Note: The commands listed here are slightly different from the commands in the Command History and Command Preview windows. The commands listed here show how you would typically execute the Xilinx programs from the system prompt, outside of the Xilinx Design Manager framework. The commands listed in the Command History and Command Preview windows reflect how Flow Engine executes Xilinx programs to fit into the Xilinx Design Manager framework.

XC4000E Command Summaries

Functional Simulation

```
qhlib work
qhmap work work
qvhcom -qhpro_syminfo src/seg7dec.vhd
qvhcom -explicit -qhpro_syminfo src/alu.vhd
pld_dve -s calc xc4000e
qhpro calc -lib work -ini quickhdl.ini
```

Basic Translation

```
[synthesize HDL modules]
pld_men2edif calc xc4000e -b a
ngdbuild -p XC4000E -uc calc_4ke.ucf calc.edif
    calc.ngd
map -p XC4003E-4-PC84 -o calc_map.ncd -oe normal
    calc.ngd calc.pcf
trce calc_map.ncd -a -o calc_map.twr
par -w -l 4 calc_map.ncd calc.ncd calc.pcf
trce calc.ncd -a -o calc.twr
ngdanno calc.ncd calc_map.ngm
ngd2edif -v mentor -w calc.nga calc.edn
bitgen calc.ncd -l -w
```

Timing Simulation

```
pld_edif2tim calc.edn
pld_dve -s calc_lib/calc xc4000e
pld_quicksim calc_lib/calc -cp -tim typ -consm
    messages
```

Further Reading

This Schematic-on-Top with VHDL Tutorial is intended to give you the information necessary to begin a Xilinx design using Mentor Graphics software. It is important to note that tools as broad and complex as Design Architect, QuickSim, QuickHDL, and QuickHDL Pro cannot be fully explained in a single tutorial. There are many different ways to use the commands in these tools, and there are also many ways to customize these applications. It is strongly recommended that you read the Mentor Graphics Design Architect,

QuickSim, QuickHDL, and QuickHDL Pro documentation as well as the *Xilinx Mentor Graphics Interface/Tutorial Guide*.

Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

Add force 7-5

B

Bus rippers 3-7
 RULE property 3-9

C

Calc_da 9-7
COMP property 3-10
Configuring your system 2-1

D

Design architect 1-1
 Tutorial
 Adding buses 9-35
 Adding labels 9-54
 Adding nets 9-36
 Adding pins to symbol 9-23
 Adding ports 9-42
 Adding text 9-26
 Assigning pin locations 9-62
 calc_3k 9-7
 Command summary 9-121, 10-49
 Completing ALU schematic 9-47
 Configuring system 9-4, 10-4
 Constraints file 9-77, 10-22
 Copying components 9-33
 Copying files 9-11, 10-8
 Creating ANDBLK2 symbol 9-22
 Creating ORBLK2 symbol 9-28
 Creating schematics for symbols 9-30
 Design description 9-20
 Entering commands 9-16
 FAST pads 9-64
 FD4CE symbol 9-52
 Files used 9-6, 10-6
 I/O Flip-Flops 9-65
 Installing 9-6, 10-5
 Labeling buses 9-43
 Men2XNF8 9-100, 10-34
 Mentor graphics variables 2-2
 Net connections 9-39
 Opening calc schematic 9-61
 Placing library components 9-51

Placing user-created components 9-49
Required software 9-4, 10-4
Saving ALU schematic 9-56
Saving calc schematic 9-66
Saving symbol 9-28
Schematic layout 9-62
Stack implementation 9-68
Using function keys 9-16
Using mouse 9-15
Viewing primitive 9-57
Viewing RPM 9-57
Viewing soft macro 9-57
XC4000 oscillator 9-61
Xilinx library elements 9-56

Design flow 9-3, 10-3
Design Manager
 translating designs to XNF format 3-27
Design manager
 Defining the interface 1-4
 Icons 1-4
 See also PLD_DMGR
Design Viewpoint Editor
 See PLD_DVE
Do file 7-4

E

Editor icon 1-6, 1-7

F

Flip-flops 7-5
FNCSIM8
 See also PLD_FNCSIM8
Functional simulation 3-16, 3-35

G

Globalresetb 7-5
Globalsetreset 7-5

I

implementation see design implementation
Incremental design 9-3
INIT property 7-5
INST property 3-10
INTERNAL property 3-10

Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

I/OB flip-flops 7-5

L

Latches 7-5

LCA 2-3

LCA file 3-35

LD_LIBRARY_PATH 2-2

M

Macros

FPGA 3-6

Soft 3-6

Manual translation

Program summary 8-11

MGC_GENLIB 2-2

MGC_HOME 2-2

MGC_LOCATION_MAP 2-2

MGC_WD 2-3

MGLS_LICENSE_FILE 2-2

Models

QuickSim II simulation 7-4

Modify property 3-13

N

Naming conventions 7-1

Nets

Analyzing 7-4

P

PINTYPE property 3-10

PLD 3-9

PLD_DMGR 1-4

Running applications 2-4

Starting 2-4

PLD_DVE

Dialog box 3-18, 3-37

PINTYPE property 3-10

PLD_Men2XNF8 9-100, 10-34

Output files 9-102, 10-36

Primitives

FPGA 3-6

Properties 3-9

FPGA

COMP 3-10

INST 3-10

INTERNAL 3-10

PINTYPE 3-10

Q

QuickPart tables 7-4

QuickPath 1-6, 3-42, 8-19

Dialog box 3-42

QuickSim II

Analyzing nets 7-4

Dialog box 3-20, 3-38

Simulation models 7-4

Tutorial

Adding traces manually 9-85

Asserting global reset 9-89

Asserting global set reset 9-90

Assigning values to clock 9-87

Design description 9-92

Opening list window 9-84

Opening trace window 9-84

PLD_TIMSIM8 9-109, 10-42

Saving waveform data 9-98

Selecting nets for simulation 9-82, 10-32

Simulating the circuit 9-92

Timsim8.log file 9-110, 10-43

Using the transcript 9-100

Viewing calc schematic 9-81, 10-28

R

Retargeting design 7-1

Rip component 3-7

S

SIMPRIMS 2-3

Simulation models 7-4

Symbols

Adding pins 9-23

T

Timing simulation 3-35

Manual translation 8-9

Tri-state (3-state) 7-6

Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

U

Unified libraries 3-5
Retargeting design 7-1

V

Viewpoint
XNF 8-13
VMH/VMD files 7-4

X

XC2000/XC3000 designs
Globalresetb 7-5
XC4000 designs
Globalsetreset 7-5
Xilinx attributes. See Properties
Xilinx libraries 7-1

Index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
