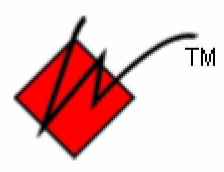
WILDSTAR™ FPDP-E I/O Daughter Card

Reference Manual

12696-0000 Revision 2.6

© Copyright 2000-2006 by Annapolis Micro Systems, Inc. All Rights Reserved. Printed and published in the United States of America. WILDFIRE™, WILDFIRE™-XL, WILDCHILD™, WILDFORCE™, WILDFORCE-XL™, WILD-ONE™, WILD-ONE™-XL, WILDTIME™, WILDCARD™, STARFIRE™, STARFIRE™ II, WILDSTAR™, WILDSTAR™-II, WILDSTAR™-E, WSDP™, WILDWARE™, WILD™, C2WILD™, CoreFire™, FIREBIRD™, and FIREBIRD™-II are trademarks of Annapolis Micro Systems, Inc. All other trademarks and registered trademarks are owned by their respective owners.


TM

ANNAPOLIS MICRO SYSTEMS, INC. - LICENSE AGREEMENT

WILDSTAR™ Family I/O Daughter Card Host Software, Models, VHDL, Examples, and Tools are supplied with a License Agreement. This License Agreement also covers the Flash content, PLD designs and FPGAs supplied with the board.

Do not install or use this product and/or break the seal on the CD-ROM until you have read and agreed to the following terms and conditions. If you agree to these terms and conditions, you should sign and return the License and Registration Certificate. You will not be entitled to support or updates until the License and Registration Certificate is received by Annapolis Micro Systems, Inc. Should you choose not to be bound by the terms and conditions of this agreement, you should promptly return this product.

YOU ARE BOUND TO THE TERMS OF THIS AGREEMENT BY BREAKING THE SEAL ON THE CD-ROM

Under the terms of this License, you:

- may make copies of the Licensed Product
- may not transfer the Licensed Product to an unlicensed party
- may modify the VHDL and Examples
- may not modify any other parts of the Licensed Product
- may not decompile, reverse assemble or otherwise reverse engineer the Licensed Product
- may run this product ONLY on an Annapolis Micro Systems, Inc. board

The Licensed Product is owned and copyrighted by Annapolis Micro Systems, Inc. You may not remove the copyright notice from the Licensed Product. You must use your best efforts to prevent any unauthorized copying of the Licensed Product.

The Licensed Product is provided "as is" without warranty of any kind including warranties for merchantability or fitness for a particular purpose. Annapolis Micro Systems, Inc. shall not be liable for any loss of profits, loss of use, interruption of business, nor for indirect, special, incidental or consequential damages of any kind whether under this agreement or otherwise.

Although Annapolis Micro Systems, Inc. does not warrant the functions contained in the Licensed Product, the medium on which the Licensed Product is furnished is warranted to be free from defects in materials and workmanship under normal use for a period of 90 days from date of delivery to you as evidenced by a copy of your receipt. Annapolis Micro Systems' entire liability to you and your exclusive remedy shall be replacement of the Licensed Product if the medium on which the Licensed Product is furnished proves to be defective.

You understand that the Licensed Product may require a license from the US Department of Commerce or other government agency before it may be taken or sent outside of the United States. You agree to obtain any required licenses before taking or sending the Licensed Product out of the United States. You will not permit the re-export of the Licensed Product without obtaining required licenses or letter of further assurance.

Annapolis Micro Systems, Inc. does not assume any liability arising out of the application or use of any product or circuit described herein.

TM

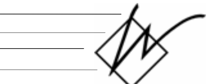


TABLE OF CONTENTS

1. ABOUT THIS MANUAL 1-1

1.1 Chapter Overview 1-1
1.2 Conventions 1-3
1.3 Icons 1-4
1.4 Key Words and Definitions 1-4

2. INTRODUCTION 2-1

2.1 Features 2-2
2.2 FPDP-E I/O Card Architecture 2-3

3. GETTING STARTED 3-1

3.1 Unpacking 3-1
3.2 Card Components 3-2

4. INSTALLING THE FPDP-E I/O CARD 4-1

4.1 Introduction 4-1
4.2 Hardware Installation 4-2
4.2.1 Installing the FPDP-E I/O Card on a VME Motherboard 4-2
4.2.2 Removing the FPDP-E I/O Card from a VME Motherboard 4-5
4.2.3 Installing the FPDP-E I/O Card on a PCI Motherboard 4-7
4.2.4 Removing the FPDP-E I/O Card from a PCI Motherboard 4-9
4.2.4.1 FPDP Cable Installation 4-10

5. TECHNICAL SUPPORT 5-1

5.1 Board Identification Numbers 5-1

6. HARDWARE REFERENCE 6-1

6.1 FPDP-E I/O Card Hardware 6-1
6.2 General Specifications 6-1
6.3 FPDP-E I/O Card Clocks 6-1
6.3.1 Configuration Register 6-2
6.3.2 UCLK 6-3
6.3.3 MCLK 6-3
6.3.4 PCLK 6-3
6.3.5 KCLK 6-3
6.3.6 XCLK 6-3
6.3.7 SKYCLK 6-4
6.3.8 FPIN_CLK 6-4
6.3.9 FPOUT_CLK 6-4
6.3.10 Frequency Parameters 6-4

7. COREFIRE™ DESIGN SUITE SUPPORT 7-1

8. VHDL MODELS REFERENCE 8-1

8.1 FPDP-E I/O Card System VHDL Model 8-1
8.1.1 VHDL Model Overview 8-2
8.1.1.1 FPDP-E I/O Card Block Diagrams 8-2
8.2 FPDP-E I/O Card PE Model 8-4
8.2.1 PE Pad Definitions and Locations 8-4
8.2.2 FPDP-E I/O Card PE Interface Components 8-4
8.2.2.1 Clock Standard Interface (Clock_Std_IF) 8-5
8.2.2.2 LAD Interfaces 8-8
***8.2.2.2.1 LAD Mux Interface (LAD_Mux_IF)** 8-8*
***8.2.2.2.2 WILDSTAR™-II LAD Standard Interface (WSII_LAD_Bus_Std_IF)** 8-10*
***8.2.2.2.2.1 LAD Bus Transactions** 8-11*
***8.2.2.2.2.2 Register Space Transactions** 8-12*
***8.2.2.2.3 WILDSTAR™-II PRO LAD Standard Interface (WSIIPRO_LAD_Bus_Std_IF)** .. 8-13*
8.2.2.3 WILDSTAR™-II PRO LAD Bus Standard Interface Reserved Registers 8-14
8.2.2.4 LAD_Mux_Reset 8-14
8.2.2.5 LAD Mux CSR PLD Interface (LAD_Mux_CSR_PLD_IF) 8-15
8.2.2.6 Mem36_Mux_IF (Priority and Fair) 8-16
8.2.2.7 FPDP Standard Interface (FPDP_Std_IF) 8-17
8.2.2.8 LED Standard Interface (LED_Std_IF) 8-18
8.2.2.9 RACEway VME Backplane Connector Standard Interface (RACEway_Std_IF) 8-18
8.2.2.10 Motherboard I/O Connectors 8-19
***8.2.2.10.1 VME Backplane Connector Standard Interface (VME_Conn_Std_IF)** 8-20*
***8.2.2.10.2 PE0 Connector Standard Interface (PE0_Conn_Std_IF)** 8-20*
***8.2.2.10.3 WILDSTAR™-II I/O Connector Basic Interface (IO_ConnWS_Basic_IO)** 8-21*
***8.2.2.10.4 WILDSTAR™-II PRO I/O Connector Basic Interface (IOConn_Type1_Basic_IO)** 8-22*
8.3 FPDP Cables 8-22
8.3.1 FPDP-E Cable Components 8-23
8.3.1.1 FPDP Transmit Cable Component 8-23
8.3.1.2 FPDP Receive Cable Component 8-24
8.4 The WILDSTAR™ Family Mux Library 8-25
8.4.1 Standard PE Templates 8-26
8.4.2 LAD_Mux Library 8-26
8.4.2.1 LAD_Mux_IF 8-27
8.4.2.2 LAD_Mux_Reset 8-28
8.4.2.3 LAD_Mux_RegFile 8-29
8.4.2.4 LAD_Mux_CRegFile 8-29
8.4.2.5 LAD_Mux_BlockRAM 8-30
8.4.2.6 LAD_Mux_BlockRAM64 8-31
8.4.2.7 LAD_Mux_Arb 8-31
8.4.2.8 Assigning the BASE to the LAD Mux Components 8-32
8.4.2.8.1 LAD_Mux_Reset 8-32
8.4.2.8.2 LAD_Mux_RegFile and LAD_Mux_CRegFile 8-33
8.4.2.8.3 LAD_Mux_BlockRAM and LAD_Mux_BlockRAM64 8-34
8.4.3 Mem_Mux Library 8-34
8.4.3.1 Memory Arbitration Schemes 8-35
8.4.3.1.1 Priority Arbitration 8-35
8.4.3.1.2 Fair Arbitration 8-36
8.4.3.2 Mem36_Mux Control 8-37
8.4.4 Programmed I/O Memory Bridge 8-39
8.4.4.1 LAD_Mem36_Bridge 8-39
8.4.4.2 Assigning the BASE to the LAD Memory Bridges 8-40
8.4.4.2.1 LAD_Mem32_Bridge 8-40
8.4.5 DMA_Mux Library 8-41
8.4.5.1.1 LAD_DMA_Write_Mux_IF 8-42

8.4.5.1.2 LAD_DMA_Read_Mux_IF 8-43
8.4.6 DMA Memory Bridges 8-45
8.4.6.1 DMA Write Bridges 8-46
8.4.6.1.1 LAD_DMA_Write_Mux_Arb 8-47
8.4.6.2 DMA Read Bridges 8-48
8.4.6.2.1 LAD_DMA_Read_Mux_Arb 8-49
8.4.6.3 Assigning the BASE to the LAD Memory Bridges 8-50
8.4.7 Host Model 8-50

9. FPDP-E I/O CARD VHDL GUIDE 9-1

9.1 VHDL Design Cycle 9-1
9.1.1 FPDP-E Template VHDL Design Files 9-1
9.1.1.1 ModelTech™ Macro Scripts 9-1
9.1.1.2 PE VHDL Template Files 9-1
9.1.1.2.1 FPDP-E I/O PE Architecture Template 9-1
9.1.1.2.2 FPDP-E I/O Connector Interface Template 9-2
9.1.1.2.3 FPDP-E Backplane Interface Template 9-2
9.1.1.3 FPDP-E Configuration Templates 9-2
9.1.2 Simulating a VHDL Design 9-3
9.1.3 Synthesizing a VHDL Design 9-3
9.1.3.1 Using Template Synplify™ Project Files 9-3
9.1.3.2 Setting up Synthesis Constraints 9-3
9.1.3.3 Running the Synplicity Synthesis Tool 9-4
9.1.4 Place-and-Routing a Design 9-4
9.1.4.1 Setting Environment Variables 9-4
9.1.4.2 Using Template Makefiles 9-5
9.1.4.3 Setting up Timing Constraints 9-5
9.1.4.4 Running the Xilinx Place-and-Route Makefile 9-5
9.1.4.5 Analyzing Design Performance 9-5
9.1.5 Transferring a PE design 9-6

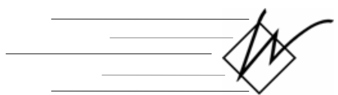
ATTACHMENT A: GENERIC QUICK REFERENCE GUIDE A-1

ATTACHMENT B: FPDP-E EXAMPLES B-1

ATTACHMENT C: STANDARD INTERFACE COMPONENTS REPLACED BY MUX COMPONENTS (WILDSTAR™ AND FIREBIRD™ ONLY) C-1

ATTACHMENT D: FPDP-E I/O CARD DLL USAGE D-1

TM



1. ABOUT THIS MANUAL

This manual provides guidance for installing and using the WILDSTAR™ FPDP-E I/O Daughter Card. This board is compatible with WILDSTAR™-E VME, FIREBIRD™ /PCI, and WILDSTAR™-II and WILDSTAR™-II PRO-series PCI and VME boards, including the single-PE PRO/ACE for VME board.

In addition, the WILDSTAR™ FPDP-E I/O Daughter Card is fully compatible with the CoreFire™ Design Suite, an FPGA design application developed by Annapolis Micro Systems, Inc.

!

CAUTION

When boards are referred to in this document, the following abbreviations are used:

- **"FPDP-E I/O Card"** stands for WILDSTAR™ FPDP-E I/O Daughter Card, unless specifically indicated otherwise.
- **"WILDSTAR™-II"** stands for WILDSTAR™-II /VME and WILDSTAR™-II /PCI boards, unless specifically indicated otherwise.
- **"WILDSTAR™-II PRO"** stands for WILDSTAR™-II PRO/VME, WILDSTAR™-II PRO/PCI, and WILDSTAR™-II PRO/ACE for VME boards, unless specifically indicated otherwise.

1.1 Chapter Overview

- Chapter 1, "**About This Manual**," outlines the conventions, icons, and key words used throughout the manual.
- Chapter 2, "**Introduction**," discusses its architecture and performance features.
- Chapter 3, "**Getting Started**," describes how to properly unpack and inspect the card.
- Chapter 4, "**Installing the FPDP-E I/O Card**," explains how to install the card and the software included with it. Installation instructions are provided for all motherboard options.

- Chapter 5, "**Technical Support**," provides information for contacting the Annapolis Micro Systems, Inc. Technical Support team.

- Chapter 6, "**Hardware Reference**," includes electrical and power specifications, clock input and sourcing options for the card.

- Chapter 7, "**CoreFire™ Support**," provides a brief overview of the CoreFire™ Design Suite, an Annapolis Micro Systems application that allows you to quickly program and debug PEs on WILDSTAR™-II motherboards in a minimal number of steps.

- Chapter 8, "**VHDL Models Reference**," provides details of VHDL interfaces and models. It also describes simulation environments, use of WILDSTAR™-II VHDL templates, record types, and processing element (PE) standard interfaces.

- In Chapter 9, "**VHDL Guide**," each phase of the VHDL design cycle is discussed, beginning with design creation and going through simulation, synthesis, place and route, and finally analysis of the design's performance.

- **Appendix A** contains a **Generic Quick Reference Guide**.

- **Appendix B** contains FPDP-E I/O Card **Euro Examples.**

- **Appendix C** contains **Standard Interface Components Replaced by Mux Components** (for WILDSTAR™ and FIREBIRD™ only).

- **Appendix D** contains information on FPDP-E I/O Card **DLL Usage.**

1.2 Conventions

A variety of text styles are used throughout the manual to call attention to specific items.

Convention	Description
Text represented as `screen display`	This typeface is used to represent displays appearing on the screen, such as: at the "`A:\`" prompt.
Text represented as `commands`	This typeface is used to represent commands that are to be entered, such as: type "`setup.`"
Keys	When specific keys are referenced, they are designated by their labels, such as "the Enter key" or "the Escape key," or they may be shown as [Enter] or [Esc]. When two or more keys are to be pressed simultaneously, the keys are linked with a plus sign (+). For example: [Ctrl] + [Alt] + [Del].

1.3 Icons

Throughout the manual, important information is highlighted with icons.

Icon	Type	Description
i	Information Note	Information Notes call attention to important features or instructions.
!	Caution	Cautions are directions that must be followed in order to avoid loss of system data and/or damage to hardware.
⚡	Warning	Warnings are directions that must be followed to ensure personal safety.

1.4 Key Words and Definitions

Some of the terms used throughout the manual are defined below.

API

Application Programming Interface.

Application Programming Interface

A set of functions coded in the C language allowing communication between an application and the board.

External I/O

External Input/Output (I/O) for the WILDSTAR™ board.

FPDP

Front Panel Data Port. *FPDP* is a trade name of Interactive Circuits and Systems, Ltd. (ICS). FPDP is an American National Standard (ANSI): ANSI/VITA 17. Data is transmitted to and from the motherboard via the FPDP I/O Card and FPDP Cable.

FPDP Cable

Ribbon cable for FPDP connection.

I/O Card 0

Designation of a WILDSTAR™ Euro I/O card installed in I/O card slot 0 of the WILDSTAR™-E /VME, WILDSTAR™-II /VME, WILDSTAR™-II PRO/VME, or WILDSTAR™-II PRO/ACE for VME board.

I/O Card 1

Designation of a WILDSTAR™ Euro I/O card installed in I/O card slot 0 of the WILDSTAR™-E /VME, WILDSTAR™-II /VME, WILDSTAR™-II PRO/VME, or WILDSTAR™-II PRO/ACE for VME board.

Motherboard

WILDSTAR™, WILDSTAR™-II, WILDSTAR™-II PRO, or FIREBIRD™ /PCI mainboard hosting the FPDP-E I/O Card.

_n

Signal name suffix denoting active low signal levels.

PCI

Peripheral Component Interconnect

PE

Processing Element.

Processing Element

A Xilinx® VIRTEX™ Field Programmable Gate Array (FPGA) comprises the main Processing Element on the FPDP-E I/O Card.

RACEway™

RACEway™ is a trademark of Mercury® Computer systems. RACEway Interlink is a standard for high-bandwidth data communication in real-time computer systems.

WILDSTAR™ *Euro I/O Daughter Card*

Eurocard form factor I/O daughter card for WILDSTAR™-E/VME, WILDSTAR™-II /VME, WILDSTAR™-II PRO/VME, and WILDSTAR™-II PRO/ACE for VME.

WILDSTAR™ *Host Software*

Host software provided for the specified WILDSTAR™ board.

WILDSTAR™*-II Host Software*

Host software provided for the specified WILDSTAR™-II or WILDSTAR™-II PRO board.

TM

2. Introduction

The FPDP-E I/O Daughter card transmits data to and from a WILDSTAR™ /VME, WILDSTAR™-E/VME, WILDSTAR™-II /VME, WILDSTAR™-II/PCI, WILDSTAR™-II PRO/PCI, WILDSTAR™-II PRO/VME, WILDSTAR™-II PRO/ACE for VME, or FIREBIRD™/PCI motherboard, or to and from another FPDP-E I/O Card. It also provides additional computational power to accommodate a number of applications and supports American National Standard FPDP specifications.

WILDSTAR™ /VME, WILDSTAR™-E/VME, WILDSTAR™-II /VME, WILDSTAR™-II /PCI, and all WILDSTAR™-II PRO motherboards are each capable of hosting up to two WILDSTAR™ FPDP-E I/O Daughter Cards. Using two I/O cards on a WILDSTAR™ motherboard provides two independent FPDPs in a single VME slot. The FIREBIRD™/PCI motherboard can host one FPDP-E card.

Each FPDP-E I/O Card contains a Virtex™-E FPGA with up to two million system gates, providing rapid processing communication between PE0 on the WILDSTAR™, WILDSTAR™-II, or WILDSTAR™-II PRO motherboard and the FPDP interface. In addition, the FPDP-E card PE has four external 32-bit memory ports, each of which can be accessed at the maximum MCLK frequency. The motherboard host has access to the FPDP-E PE via the LAD Bus.

Each FPDP-E I/O Card has five user-programmable LEDs, with two of these available on the front panel. The front panel LEDs are used for debugging and as functional indicators.

CoreFire™ can be used to create FPGA designs for the FPDP-E I/O Card. The CoreFire™ Design Suite, a design tool developed by Annapolis Micro Systems, Inc., makes it possible to create designs in a fraction of the time required for a conventional VHDL-based control flow approach. CoreFire™ can be used to program the I/O card PE mounted on a compatible Annapolis Micro Systems, Inc. motherboard.

2.1 Features

- When installed on a WILDSTAR™, WILDSTAR™-II, or WILDSTAR™-II PRO-series motherboard, the assembly does not require an additional VME slot.

- Up to an additional two million system gates per FPDP-E I/O Card for a total of four million per slot

- Up to 4 MBytes per port for a total of up to 16 MBytes per FPDP-E I/O Card. On Revision B of the FPDP-E I/O card, up to 8 Mbytes per port for a total of up to 32 Mbytes per card.

- Front panel FPDP connection per I/O card

- Front Panel Data Port connector for use as a test access port or for data transmission

- VME backplane connection of up to 96 bits

- Capable of supporting optional RACEway™ interface

- Capable of Single RACEway, Dual RACEway™, Single RACE™++, Dual RACE™++, and SKY Channel support (WILDSTAR™/VME and -E/VME only)

- Virtex™-E support

2.2 FPDP-E I/O Card Architecture

The FPDP-E I/O Card architecture is shown in Figure 2-3 below. The I/O PE is configured by the user. The FPDP-E I/O Card communicates to the motherboard via the external I/O connectors on the left side of the block diagram. The FPDP and the LED display are shown on the right side of the block diagram.

Figures 2-4 through 2-6 display the FPDP-E I/O Card architecture as it connects with WILDSTAR™, WILDSTAR™-II, and FIREBIRD™ motherboards.

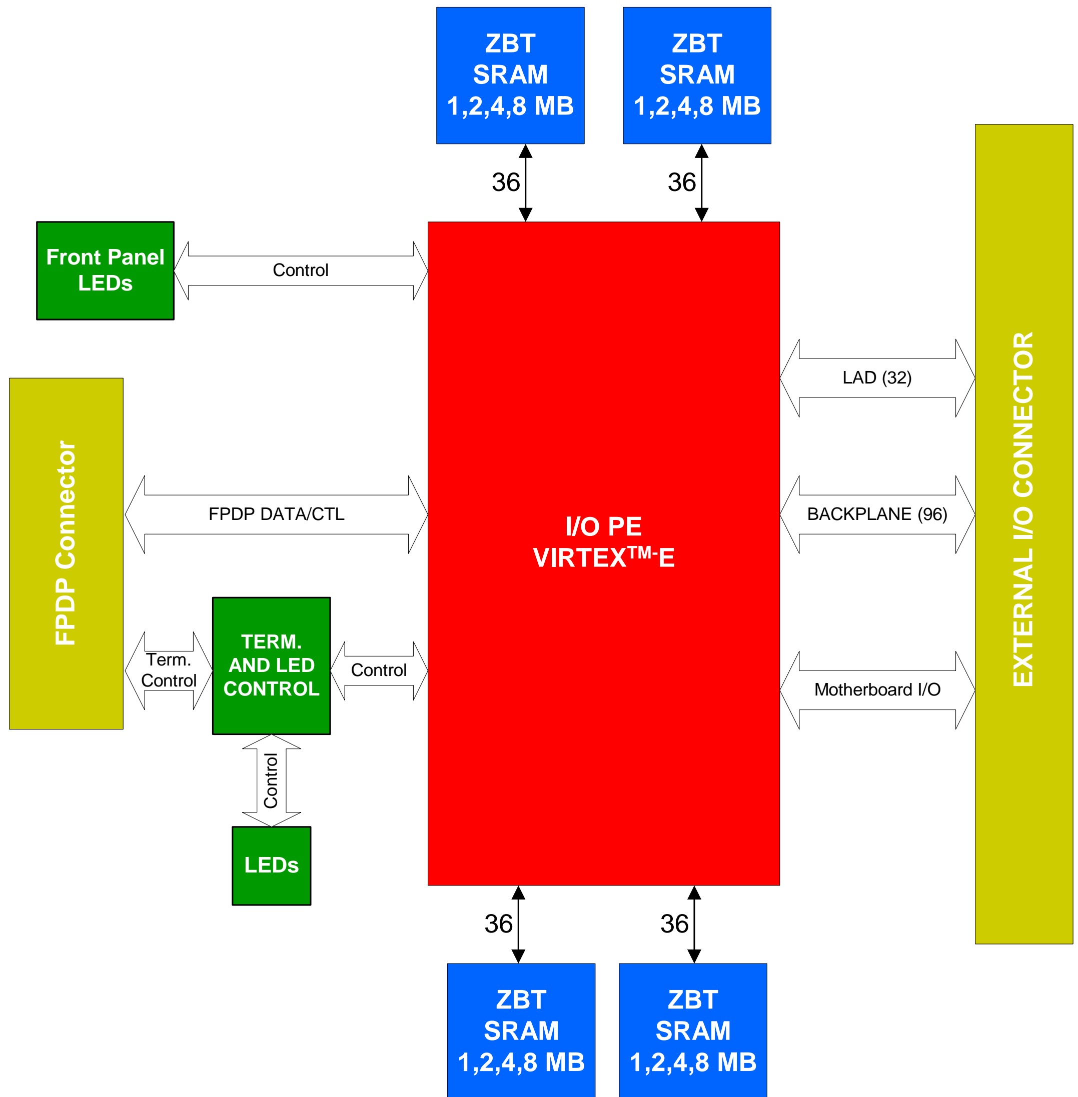
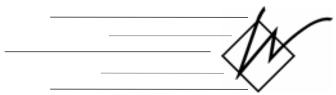


Figure 2-1: FPDP-E I/O Card Block Diagram

TM



3. Getting Started

3.1 Unpacking

The FPDP-E I/O Card is shipped in a static sensitive pack.

!

CAUTION
Before removing the card from the static pack, be sure that you are properly grounded against static electricity. Use of a ground strap for this purpose is highly recommended since static discharge to the FPDP-E I/O Card could damage its sensitive components.

Ensure that the following items are included with the card:

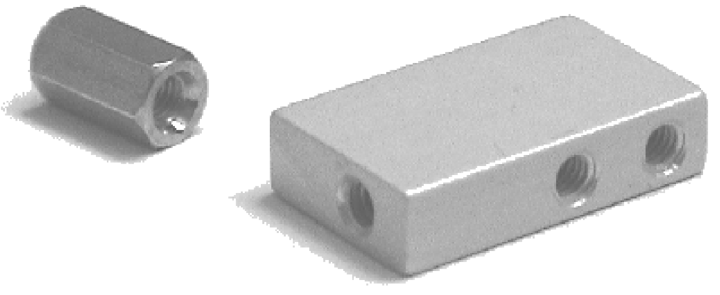
DOCUMENTATION
- Reference Manual
- Performance Plot

CD-ROMs
- FPDP-E I/O Card Documentation CD-ROM, containing PDF (portable documentation file) of this manual.
- FPDP-E I/O Card VHDL CD-ROM, which includes VHDL models and examples.

HARDWARE (illustrated below)

For VME Motherboards:
- 12 screws
- 6 standoffs

For PCI Motherboards:
- 8 screws
- 2 standoffs
- Two mounting blocks
- Back plate



Standoff **Mounting Block**

(Magnified to show detail)

When handling the card, grasp it carefully by its sides. Avoid touching any of the components on the card's surface, as they are sensitive and can be easily damaged. Visually inspect it for damage that may have occurred during shipping. If there is any apparent damage to the card or any items missing from the shipment, contact Annapolis Micro Systems, Inc. using the information provided in Chapter 5 of this manual.

3.2 Card Components

Figures in the following pages illustrate major FPDP-E I/O Card part locations on component and solder sides. Information on the location and significance of the FPDP-E I/O Card LEDs is also provided later in this chapter.

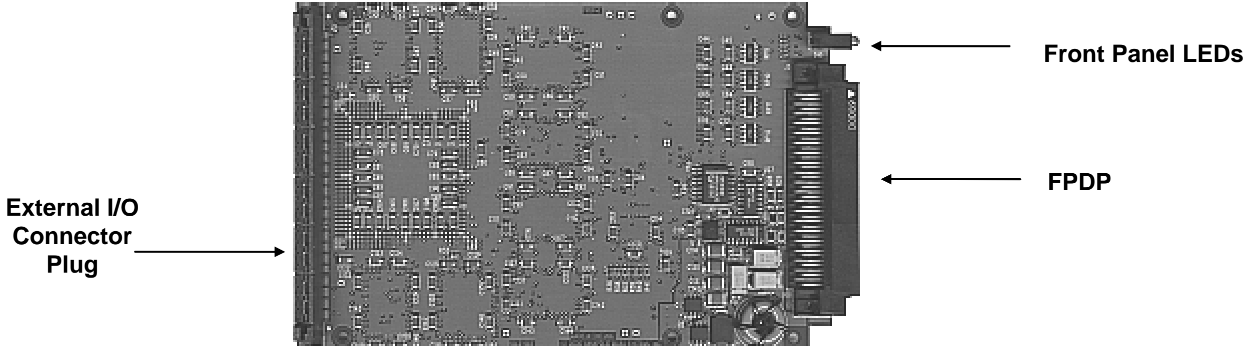
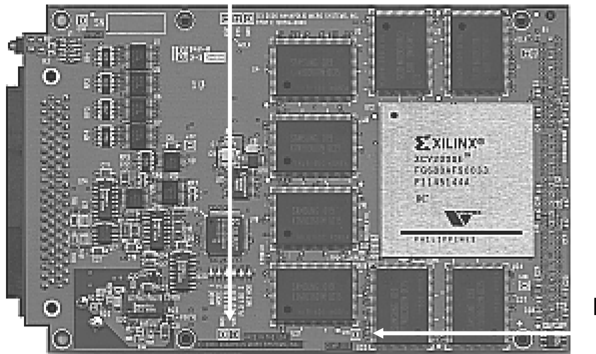


Figure 3-1: FPDP-E I/O Card (Solder Side)



PE Done
LED

Figure 3-2: FPDP-E I/O Card (Component Side)

PLD User
LEDs

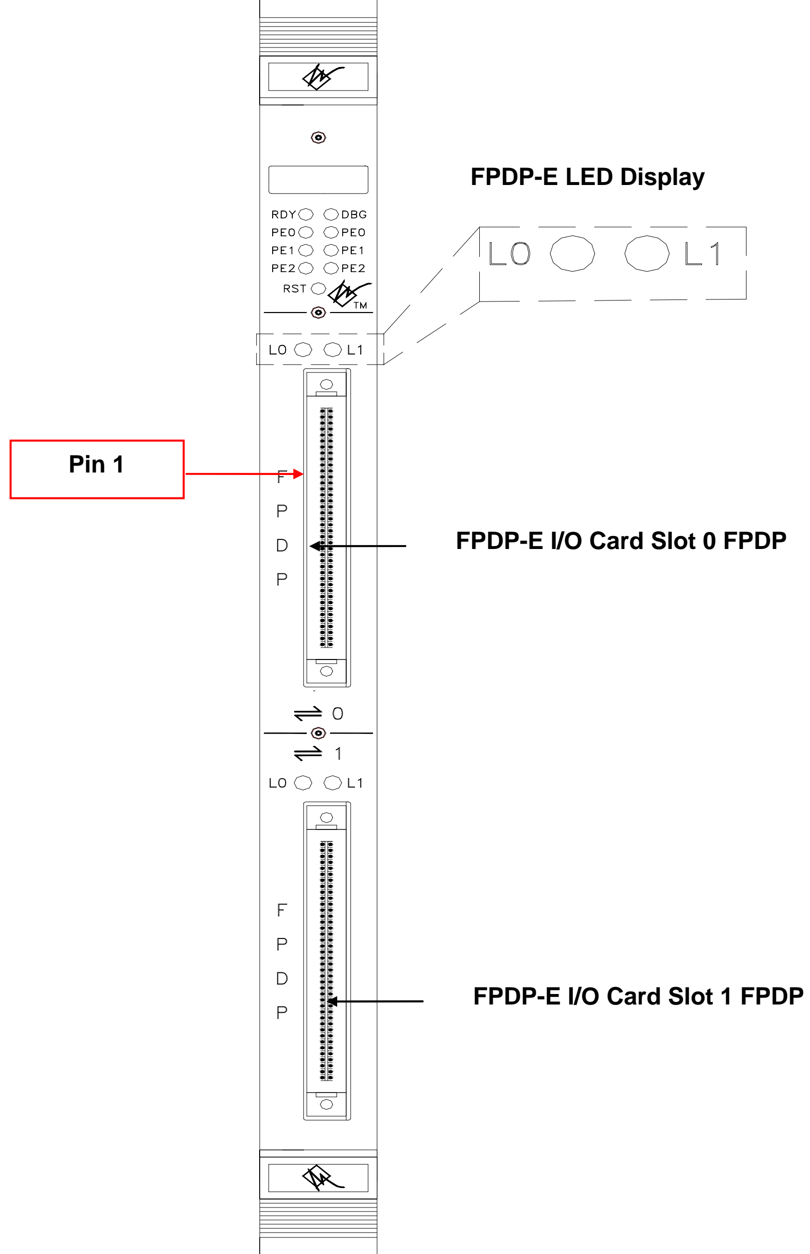


Figure 3-3: FPDP-E I/O Card Front Panel for WILDSTAR™

i

INFORMATION NOTE
The front panel shown in Figure 3-3 has double FPDP-E slots, however, boards are typically shipped with a double WSDP front panel.

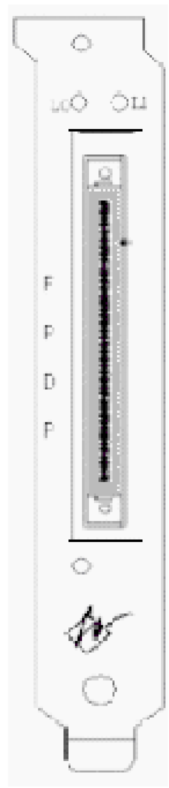


Figure 3-4: FPDP-E Card Back Plate for PCI

Table 3-1: FPDP-E I/O Card LED Definition

LED Name	Location	LED Label	LED Color	LED Default Power on State	User Configurable	LED Definition
LEDS_x_n(0)	Front Panel	U0	Green	OFF	Yes	PE User Led 0
LEDS_x_n(1)	Front Panel	U1	Green	OFF	Yes	PE User Led 1
PE Done	Component Side	D4	Green	OFF	No	PE Programmed
PLD USER 2	Component Side	D1	Green	OFF	Yes	PLD User Led 2
PLD USER 1	Component Side	D3	Red	OFF	Yes	PLD User Led 1
PLD USER 0	Component Side	D2	Green	OFF	Yes	PLD User Led 0
MODE 1	Component Side	D3	Red	OFF	No	FPDP MODE 1 Active

LED Name	Location	LED Label	LED Color	LED Default Power on State	User Configurable	LED Definition
MODE 2	Component Side	D4	Red	OFF	No	FPDP MODE 2 Active
METHOD	Component Side	D8	Red	OFF	No	FPDP METHOD Active

4. INSTALLING THE FPDP-E I/O CARD

4.1 Introduction

This chapter provides diagrams and a series of steps for installing the FPDP-E I/O Card. Two sets of installation instructions are included here—one for VME motherboards (including PRO/ACE for VME), and one for PCI motherboards.

i

INFORMATION NOTE
The PDF will be installed during Host Software installation or can be accessed from the /doc directory of the CD-ROM without installation.

4.2 Hardware Installation

4.2.1 Installing the FPDP-E I/O Card on a VME Motherboard

To install the FPDP-E I/O Card on a WILDSTAR™/VME, WILDSTAR™-E/VME, WILDSTAR™-II /VME, or WILDSTAR™-II PRO-series motherboard, follow the steps listed below:

INFORMATION NOTE
Follow Steps 1-5 if the motherboard is installed in the host computer. Once the board has been removed from the computer, follow Steps 6-15.

1. Connect the anti-static ground strap.
2. Shut down the host system and power off.
3. If necessary, remove the cover to the VME chassis in order to gain access to the VME board.
4. Use the VME board ejectors to release the board from the slot.
5. Remove the VME board from the chassis.
6. Place the VME board, component-side up, on a flat electrostatic-protected surface.
7. Remove the FPDP-E I/O Card from its static-sensitive pack. Hold the FPDP-E I/O Card by its edges and place it gently on a flat, electrostatic-protected surface.
8. Attach six, 11mm standoffs (included with the card package) to the solder side of the FPDP-E I/O Card, using six, M2.5 x 4mm screws to secure the standoffs. See Figure 4-2 for standoff locations, and Figure 4-3 for screw locations.
9. Carefully slide the connectors through the proper cutouts of the VME front panel. See Figure 4-1.
10. Align the connector plug on the solder side of the FPDP-E I/O Card with the connector receptacle on the VME board.
11. Install the FPDP-E I/O Card by pressing gently above the connectors until the connectors are firmly seated.
12. Secure the FPDP-E I/O Card to the VME board with six, M2.5 x 4mm screws through the solder side of the motherboard.
13. Reinstall the VME board with the mounted FPDP-E I/O Card into the computer. Refer to the installation section of the VME motherboard reference manual, if necessary.

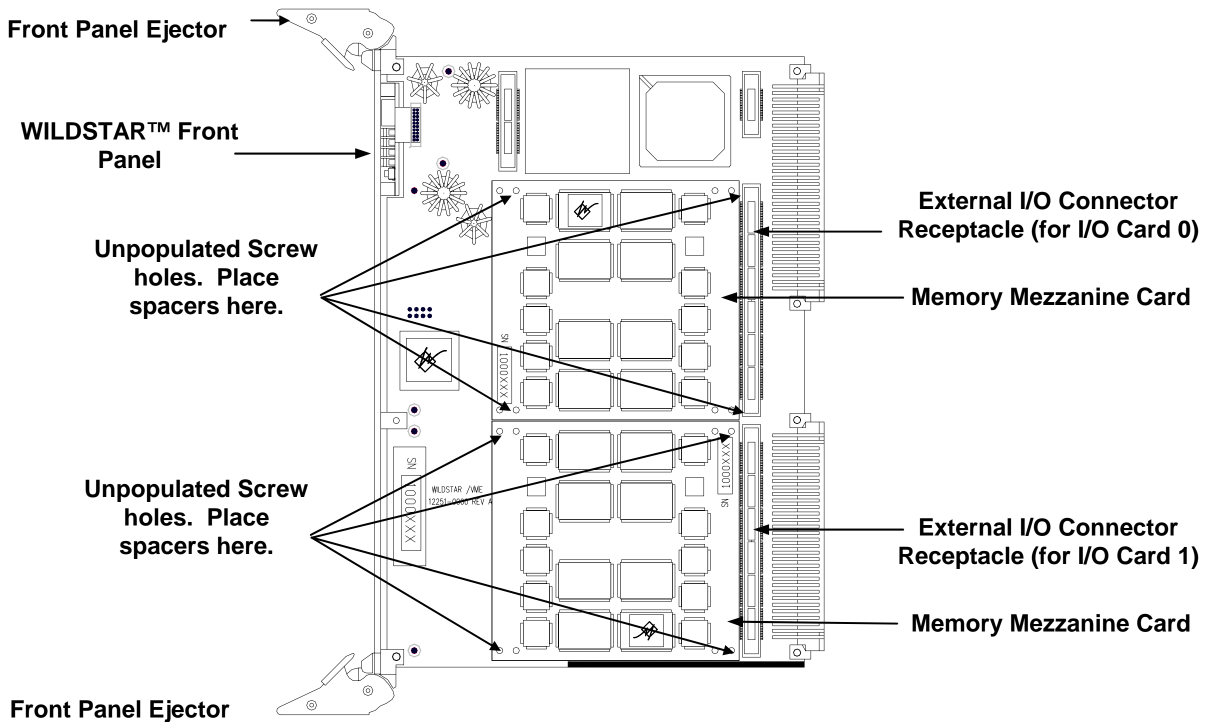


Figure 4-1: WILDSTAR™/VME Board (Component Side)

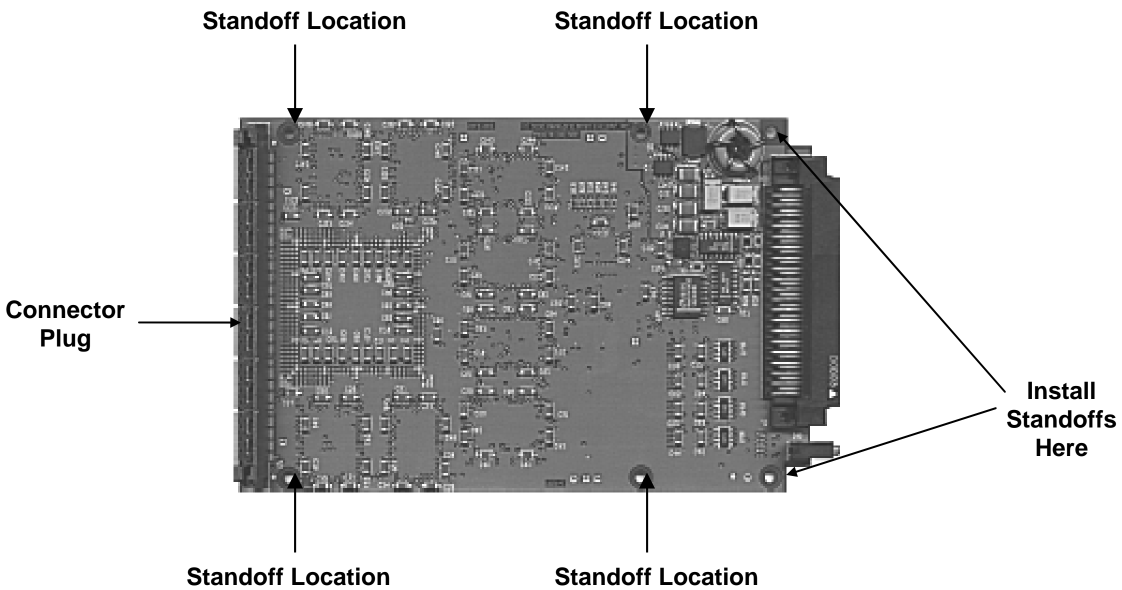


Figure 4-2: FPDP-E I/O Card (Solder Side)

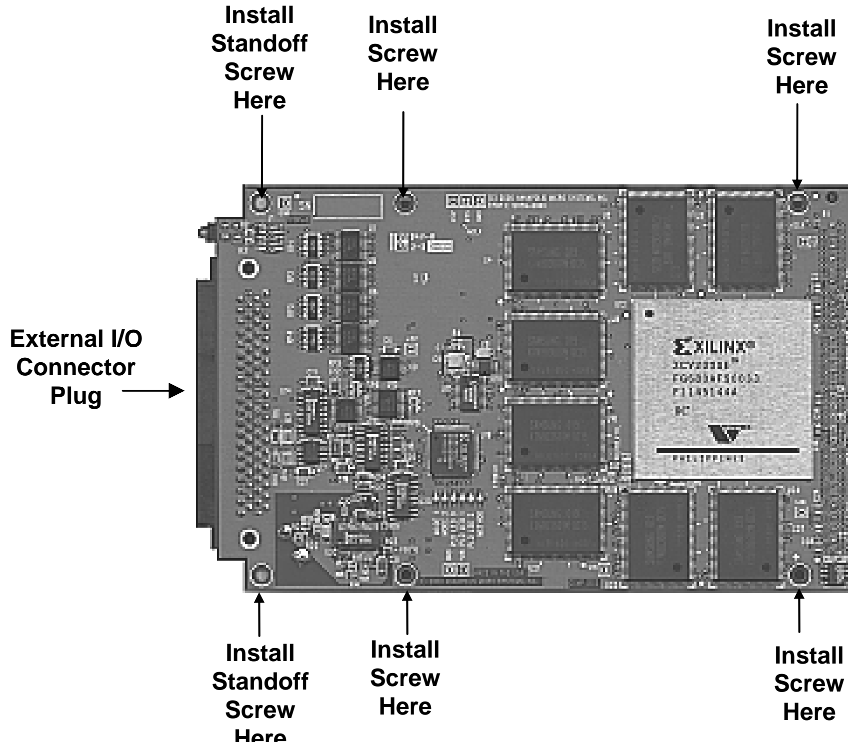


Figure 4-3: FPDP-E I/O Card (Component Side)

4.2.2 Removing the FPDP-E I/O Card from a VME Motherboard

To remove FPDP-E I/O Card from the VME motherboard, follow the steps listed below.

1. Connect the ground strap to yourself.
2. Shut down the host system and power off.
3. If necessary, remove the cover to the system chassis in order to gain access to the VME board.
4. Remove any installed cables.
5. Use the VME ejector handles to release the board from the slot.
6. Remove the VME board from the chassis.
7. Place the VME board, component-side up, on a flat electrostatic-protected surface.
8. Remove the four, M2.5 x 4mm screws attaching the FPDP-E I/O Card to the VME board. The screws will be accessible on the solder side of the VME board. The FPDP-E I/O Card is attached directly to the VME board with four standoffs and screws.
9. Using a gentle rocking motion, unseat the card connector plug from the VME board connector receptacle. Remove the FPDP-E I/O Card.
10. Carefully store the FPDP-E I/O Card in a static-sensitive pack. Reserve the standoffs and screws for future use.

To reinstall the VME board, refer to the installation section of the motherboard's reference manual.

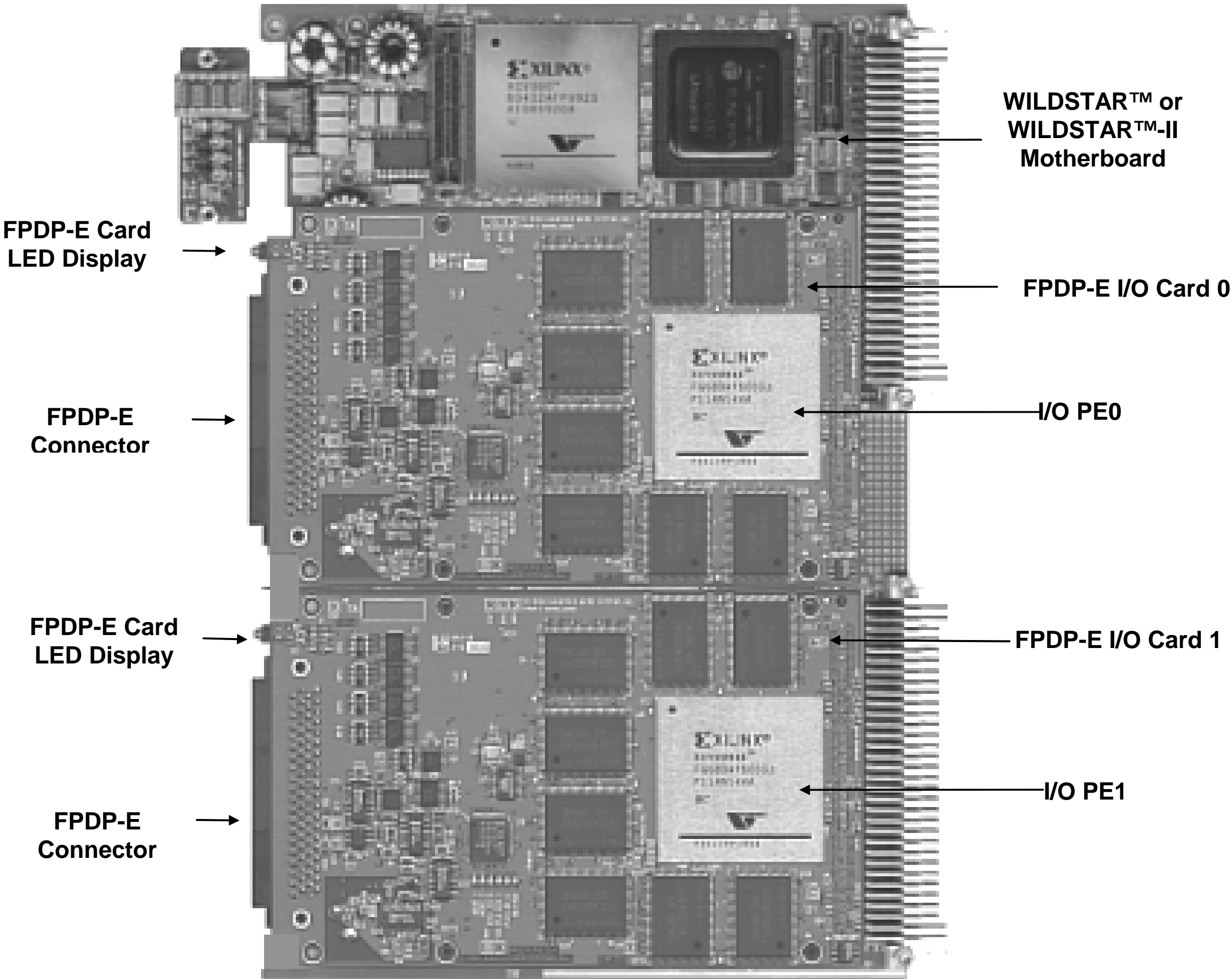


Figure 4-4: WILDSTAR™/VME with FPDP-E I/O Card 0 installed in I/O Slot 0 and FPDP-E I/O Card 1 installed in I/O Slot 1

4.2.3 Installing the FPDP-E I/O Card on a PCI Motherboard

To install the FPDP-E I/O Card on a FIREBIRD™ /PCI, WILDSTAR™-II /PCI, or WILDSTAR™-II PRO/PCI motherboard, follow the steps below:

i

INFORMATION NOTE
Follow Steps 1-5 if the motherboard is installed in the host computer. Once the board has been removed from the computer, follow Steps 6-15.

1. Connect the anti-static ground strap.
2. Shut down the host system and power off.
3. If necessary, remove the cover to the chassis to gain access to the motherboard.
4. Remove the upward-facing screw securing the steel back plate of the motherboard to the chassis, then lift out the board and the attached back plate.
5. Remove the two screws holding the standard back plate to the motherboard. Lay the back plate aside.
6. Place the motherboard component-side up on a flat electrostatic-protected surface.
7. Remove the card from its static sensitive pack. Hold the card by its edges and place it carefully on a flat electrostatic protected surface.
8. Attach two standoffs (included with the card) to the component side of the motherboard by securing them with two, M2.5 x 4mm screws. Screw them in from the solder side.
9. Attach two mounting blocks to the component side of the motherboard by securing them with two, M2.5 x 4mm screws through the motherboard's solder side. The off-center screw hole in the side of each mounting block should be closest to the motherboard.
10. Align the connector plug on the solder side of the FPDP-E I/O Card with the connector receptacle on the motherboard.
11. Press the card gently against the connector receptacle until the connector is completely seated.
12. Install six, M2.5 x 4mm screws (packaged with the card assembly) through the component side of the FPDP-E I/O Card. Four of these screws will go into the standoffs, while two will go into the tops of the mounting block screw holes.

13. Locate the back plate provided with the FPDP-E I/O Card and slide it over the cable receptacle. Attach it to the mounting blocks using two of the screws provided.

Reinstall the motherboard with the mounted FPDP-E I/O Card into the computer. Refer to the installation section of the appropriate motherboard reference manual, if necessary.

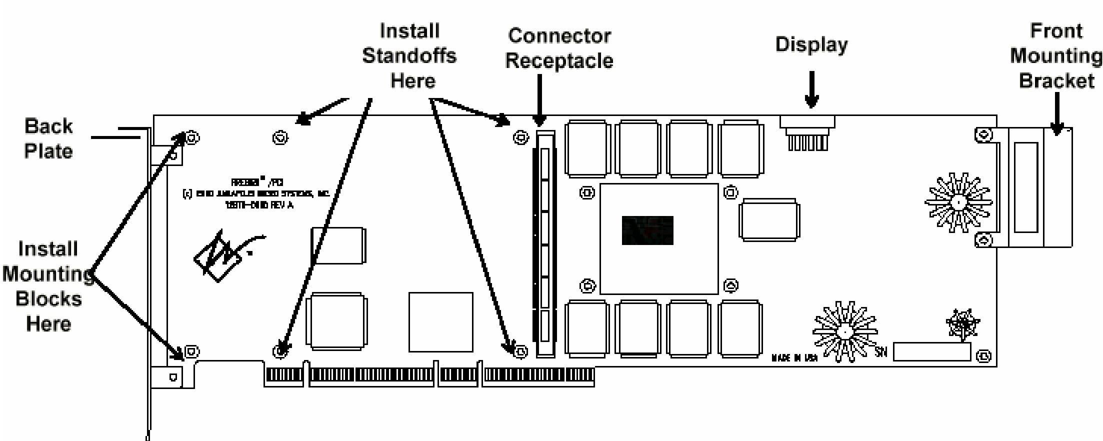


Figure 4-5: FIREBIRD™/PCI Board (Component Side)

4.2.4 Removing the FPDP-E I/O Card from a PCI Motherboard

To remove the FPDP-E I/O Card from a FIREBIRD™ /PCI, WILDSTAR™-II /PCI, or WILDSTAR™-II PRO/PCI motherboard, follow the steps below:

1. Connect the ground strap to yourself and power off the host computer.
2. If necessary, remove the cover to the system chassis in order to gain access to the motherboard.
3. Carefully remove any installed cables.
4. Remove the upward-facing screw securing the steel back plate of the PCI motherboard to the chassis, then lift out the board and the attached back plate.
5. Remove the two screws holding the back plate to the mounting blocks on the motherboard. Lay the back plate aside.
6. Place the motherboard component-side up on a flat electrostatic protected surface.
7. Remove the six screws attaching the card to the motherboard.
8. Using a gentle rocking motion, unseat the FPDP-E I/O Card connector plug from the motherboard connector receptacle. Remove the card.
9. Remove the four standoffs from the motherboard.
10. Also, remove the two mounting blocks from the motherboard.
11. After removal, store the card in a static-sensitive pack. Reserve the installation hardware for future use.
12. Replace the standard back plate to the motherboard and secure with two screws.

To reinstall the PCI motherboard, refer to the installation section of the appropriate motherboard reference manual.

4.2.4.1 FPDP Cable Installation

The FPDP on the FPDP-E I/O Card is presented through the front panel of the WILDSTAR™ board or the back plate of the FIREBIRD™ board (see Figures 3-3 and 3-4 respectively). The FPDP cable, illustrated in Figure 4-6, is plugged into the FPDP-E I/O Card.

INFORMATION NOTE
The FPDP cable **is not included** with the FPDP-E I/O Card.

The FPDP cable plugs are keyed, and can only be plugged in one way.

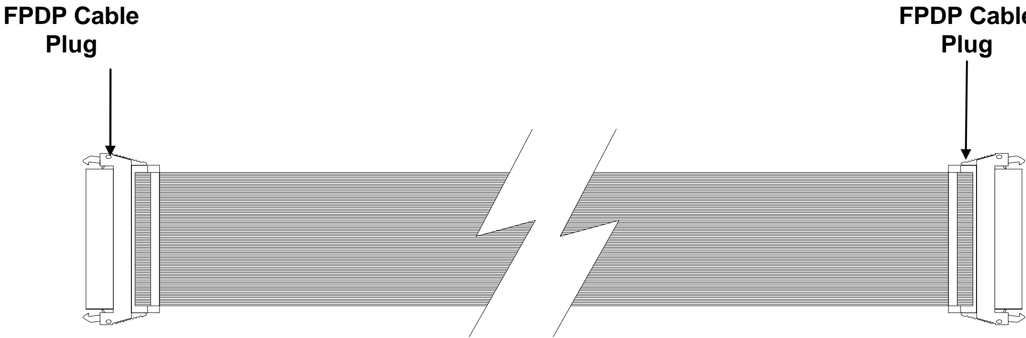
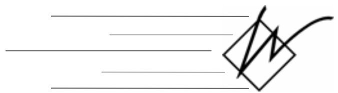


Figure 4-6: FPDP Cable



5. Technical Support

If you have any questions about installing, programming, using, or maintaining your WILDSTAR™ I/O card, please call the WILDSTAR™ Technical Support team at (410) 841-2514, fax at (410) 841-2518, or send e-mail to wftech@annapmicro.com. Our web site address is http://www.annapmicro.com.

The suggestions listed below will help us respond to your questions more quickly.

5.1 Board Identification Numbers

Each Annapolis Micro Systems board is prominently labeled with three unique codes: the *Product Configuration Code* (PCC), the *Serial Number* (SN), and the *Revision Level Code* (RLC). You can also find these codes by installing the board and running *wsinst.exe* from the host software CD.

- The **Product Configuration Code (PCC)** identifies PE type, memories, clocking, and other options selected for the particular board.

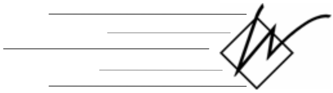
- The **Serial Number (SN)** is a unique number identifying each board.

- The **Revision Level Code (RLC)** includes information about revisions and engineering modifications made to the board.

When contacting Annapolis Micro Systems with board-related questions, please include these codes in your query, as well as the information listed below:

- Board operating system
- Host software version
- Host platform
- Host OS
- VHDL version number
- API version
- Driver versions
- CoreFire™ version being used (if applicable)

TM



6. HARDWARE REFERENCE

6.1 FPDP-E I/O Card Hardware

This chapter contains hardware reference information including switching and DC electrical specifications, I/O connector pinouts, and power consumption specifications for the FPDP-E I/O Card.

6.2 General Specifications

Table 6-1 specifies the physical dimensions and operating range for the FPDP-E I/O Card.

Table 6-1: FPDP-E I/O Card Specifications

Physical Dimensions:	Length: 143.51mm/5.65 in Width: 91.44mm/3.6 in Thickness: 2.441mm/.062 in Weight: 91g
Operating Range:	Temperature: 0º to 70ºC

6.3 FPDP-E I/O Card Clocks

The FPDP-E I/O Card has seven main clocks that can be used by the I/O PE—MCLK, PCLK, UCLK, KCLK, XCLK, SKYCLK, and FPINCLK. Figure 6-1 illustrates the architecture of the FPDP-E I/O clock scheme. The motherboard sources MCLK, PCLK, UCLK, and KCLK.

The P2 connector on the VME backplane line sources XCLK or SKYCLK on WILDSTAR™ and the FPDP connector sources FPINCLK. In addition to the five clock lines sourced by the motherboard, UCLK_SRC and MCLK_SRC are sourced by the FPDP-E I/O Card to the motherboard and FPOUTCLK is sourced to the FPDP connector. These signals allow MCLK and UCLK to be synched to an external clock.

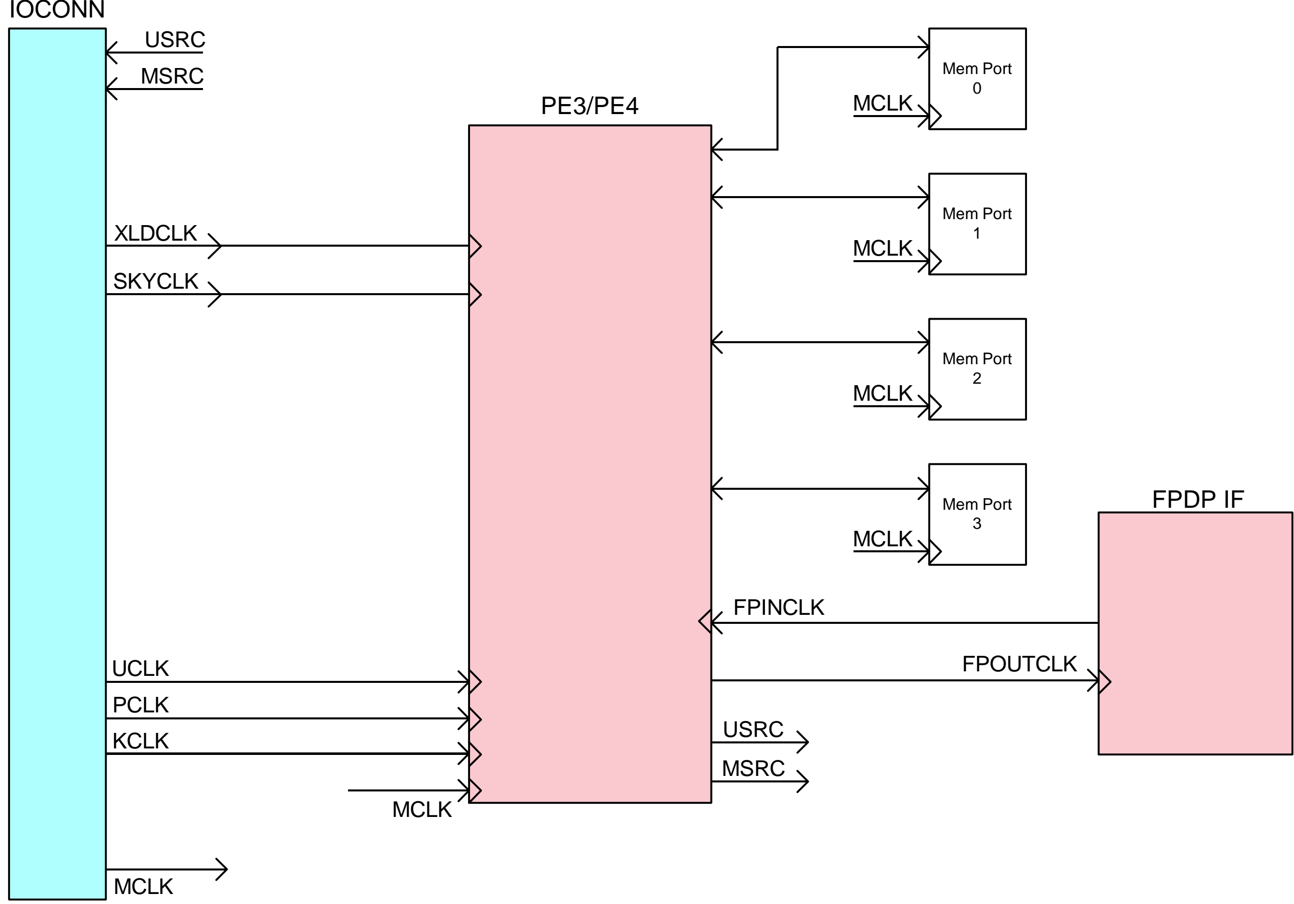


Figure 6-1: FPDP-E I/O Card Clock Structure

6.3.1 Configuration Register

The Clock Configuration Register controls the clocks on the FPDP-E I/O Card. The I/O PE always receives KCLK and MCLK, but has two pins for other clocks. The source configurable clocks are configured by the Clock Configuration Register. All of this is done with a 24-bit register inside the Clock Configuration Register which is set by the I/O PE via a serial data line. The following is a description of the bits in this 24-bit register:

Table 6-2: FPDP-E I/O Card Configuration Register

Bit Name	Description	Bit Number	Access	Init Value
PGOOD	1 = 1.8V Power Good	13	R	1
UCLKMSTR	1 = I/O Card is sourcing UCLK	12	R	0
MCLKV33ERR	1 = MCLK PLL Power Good	11	R	1
ZZ3	1 = Power Sleep Mode Memory Port 3 Enable	10	R/W	0
ZZ2	1 = Power Sleep Mode Memory Port 2 Enable	9	R/W	0
ZZ1	1 = Power Sleep Mode Memory Port 1 Enable	8	R/W	0
ZZ0	1 = Power Sleep Mode Memory Port 0 Enable	7	R/W	0
~TTLCLKEN	0 = ENABLE TTL FPCLKOUT, 1 = DISABLE TTL	6	R/W	0
CLKINSEL	1 = TTL FPCLKIN, 0 = PECL FPCLKIN (no effect if FPDP/TM)	5	R/W	0
PIODIRSEL1	0 = Transmit PIO1, 1 = Receive PIO1	4	R/W	0
PIODIRSEL2	0 = Transmit PIO1, 1 = Receive PIO2	3	R/W	0
LED2	1=turn LED2 on	2	R/W	0
LED1	1=turn LED1 on	1	R/W	0
LED0	1=turn LED0 on	0	R/W	0

6.3.2 UCLK

UCLK is a user configurable clock. It is asynchronous to MCLK, PCLK, and KCLK. The source of UCLK is selectable via the WILDSTAR™ Host Software between the WILDSTAR™ programmable oscillator, external I/O card 0, external I/O card 1, and the backplane.

6.3.3 MCLK

MCLK is synchronous to PCLK, and is asynchronous to UCLK and KCLK. The source of MCLK is selectable via the WILDSTAR™ Host Software between the WILDSTAR™ programmable oscillator, external I/O card 0, external I/O card 1, and the backplane.

6.3.4 PCLK

PCLK is the primary processing clock for the PE. PCLK is synchronous to MCLK, and is asynchronous to UCLK and KCLK. PCLK is a derivative of MCLK and the divisor is programmable by the WILDSTAR™ Host Software.

6.3.5 KCLK

KCLK is the Local Address Data Bus (LAD Bus) clock. KCLK is asynchronous to UCLK, MCLK, and PCLK. The PE uses this clock to interface to the PCI Controller for host access via the LAD bus.

6.3.6 XCLK

XCLK comes via the VME backplane on WILDSTAR™. When the FPDP-E I/O Card is plugged into the second I/O daughter card slot, this signal comes from A01 on the VME P2 connector. This pin is the signal "XCLKI" when plugged into a RACEway backplane. If plugged into the first I/O daughter card slot, this line comes from B01 on the P0 VME connector.

6.3.7 SKYCLK

SKYCLK comes via the VME backplane on WILDSTAR™. When the FPDP-E I/O Card is plugged into the second I/O daughter card slot, this signal comes from A32 on the VME P2 connector. This pin is the signal "SKYCLK" when plugged into a RACEway backplane. If plugged into the first I/O daughter card slot, this line comes from E19 on the P0 VME connector.

6.3.8 FPIN_CLK

This is the FPDP clock coming from the FPDP connector. The FPDP-E I/O Card must not be in master mode (bit 20 above) for this clock to be used.

6.3.9 FPOUT_CLK

This clock becomes the FPDP clock (STROB, PSTROB, PSTROB_n) when the FPDP-E card is in master mode (bit 20 above).

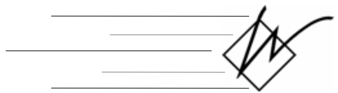
6.3.10 Frequency Parameters

Table 6-3 describes each clock, its frequency ranges, and its destination on the FPDP-E I/O Card.

Table 6-3: FPDP-E I/O Card Clock Description

Clock	Full Name	Clock Source	Frequency Range	Destination
UCLK	User Clock	WILDSTAR™/VME	320 KHz to 100 MHz***	Optional on I/O PE, Memory and FPDP clock
MCLK	Memory Clock	WILDSTAR™/VME	*15 MHz to 100 MHz***	I/O PE; Optional on Memory
PCLK	PE Clock	WILDSTAR™/VME	15 MHz to 100 MHz***	Optional on I/O PE, Memory and FPDP clock
KCLK	System Clock	WILDSTAR™/VME	33 MHz or 66 MHz	I/O PE; Optional on Memory and FPDP clock
XLDCLK	VME Backplane Generated Clock	WILDSTAR™/VME	0 MHz to 100 MHz	I/O PE
SKYCLK	VME Backplane Generated Clock	WILDSTAR™/VME	0 MHz to 100 MHz	I/O PE
MCLK_SRC	External User Clock Source	FPINCLK, XLDCLK, or SKYCLK	**7.5 MHz to 50 MHz	WILDSTAR™/VME
UCLK_SRC	External User Clock Source	FPINCLK, XLDCLK, or SKYCLK	0 MHz to 100 MHz	WILDSTAR™/VME

* When using Memory Cards the minimum is 25 MHz.
** When using Memory Cards the minimum is 12.5 MHz.
*** When using a WILDSTAR™-E motherboard, the frequency range increases to 133MHz.



7. COREFIRE™ DESIGN SUITE SUPPORT

The CoreFire™ Design Suite, an FPGA programming tool produced by Annapolis Micro Systems, allows you to create PE designs and load them onto WILDSTAR™ and FIREBIRD™ boards and I/O cards in a fraction of the time required for conventional VHDL. With the CoreFire™ Application Builder, you simply create dataflow diagrams by choosing cores from the available libraries, dropping them into an editing field, and connecting their ports. The CoreFire™ Design Suite includes the Application Builder and an Application Debugger, the latter being useful for monitoring dataflow through a diagram.

When you build a diagram, CoreFire™ generates the hardware design and a corresponding Java-based software model (shown below). The software model functions as an interface between the board hardware and the user application. The CoreFire™ Application Debugger also relies on the software model to access trace points that have been built into the design.

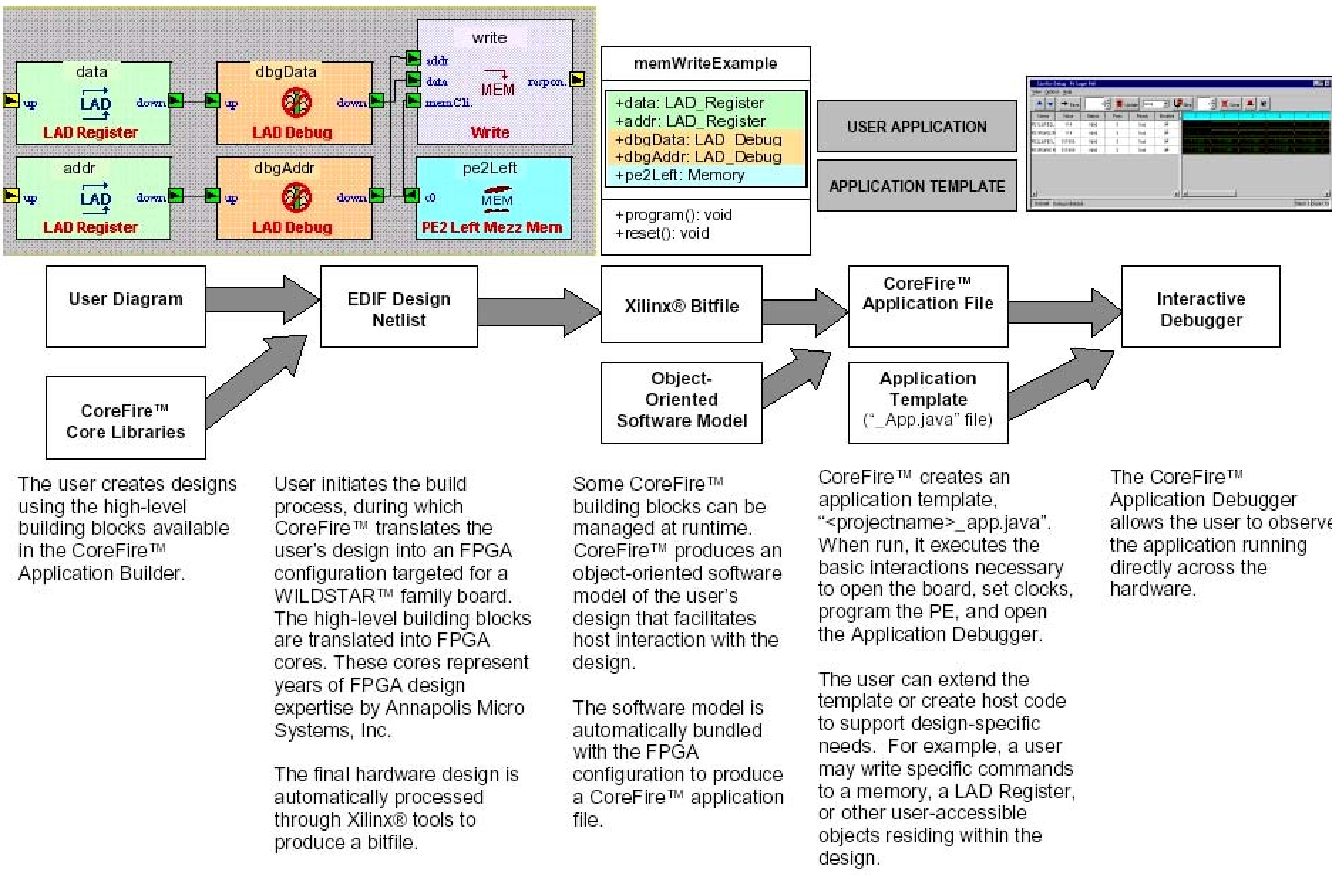


Figure 7-1: CoreFire Design Flow Model

CoreFire™ "cores" are the building blocks used for creating dataflow diagrams. Cores are grouped in libraries according to function, with libraries dedicated to

specific PEs on WILDSTAR™ motherboards and I/O cards manufactured by Annapolis Micro Systems, Inc. You can also build customized cores, called Core Macros, using cores from the libraries (excluding board-specific libraries).

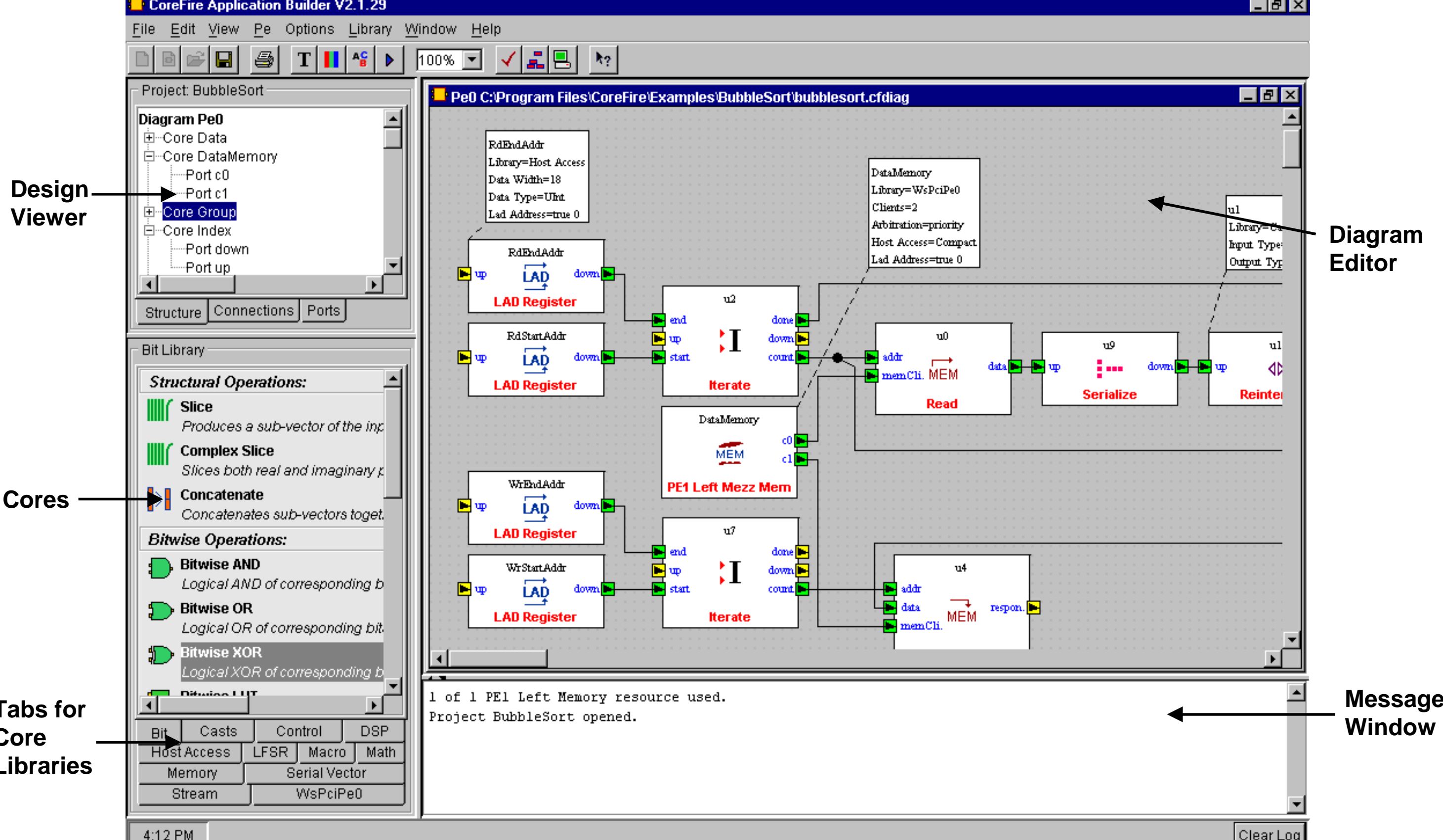


Figure 7-2: CoreFire™ Application Builder—Main Interface

Below are brief descriptions of the CoreFire™ libraries, including those specific to the FPDP-E I/O Card. Separate FPDP-E I/O Card libraries exist to accommodate the I/O card on a specific WILDSTAR™ or FIREBIRD™ motherboard.

Bit Library

Cores in the Bit Library perform various operations on individual bits and bit vectors. For instance, the Concatenate core assembles bits into larger fields, the Slice core disassembles bit fields, and the Bitwise, Buswide, and Manipulation cores perform a variety of Boolean operations.

Math Library

Math Library cores perform operations involving unsigned, signed, and floating point arithmetic.

Casts Library

Cores in the Casts Library perform data type conversions. They transform data types in two ways, by *Reinterpreting* or *Converting*. *Reinterpreting* maintains the binary representation of a given set of data, but changes its numerical value. *Converting* maintains the numerical value of data, but changes its binary representation.

Control Library

Control Library cores are used for creating iterative structures, for selectively routing and merging streams of data, and for multiplexing arithmetic operations.

Stream Library

Cores in the Stream Library perform operations on data without actually changing its value. These cores reorder, control, and group data, among other similar functions.

Serial Vector Library

Cores in the Serial Vector Library perform a wide array of data-serializing operations, including histograms and bubble sorts.

DSP Library

Cores in the Digital Signal Processing (DSP) Library perform numerous digital signal-related operations. Cores such as the FIR Filter are used to manipulate signals in order to enhance their distinguishing characteristics.

LFSR Library

The Parallel LFSR (Linear Feedback Shift Register) Library is used for generating a random stream of values starting from an initial seed value.

Host Access Library

Cores in the Host Access Library are those that can be read or written to from the host.

Memory Library

Memory Library cores, which are not specific to any PE, allow you to read and write to memory. Block RAM and other cores in the library provide small memory resources, while the Content Addressable Memory (CAM) cores make it possible to locate data in memory and determine its availability.

Macro Library

Macro libraries are created by the user for storing custom-made Core Macros.

WsVmePe0, WsVmePex Libraries

Cores in these libraries are dedicated to designs for PEs residing on a WILDSTAR™ / VME motherboard. Each library includes cores for memory, channels between PEs, and channels between PEs and I/O cards.

WsPciPe0, WsPciPex Libraries

Cores in these libraries are dedicated to designs for PEs residing on a WILDSTAR™ / PCI motherboard. Each library includes cores for memory, channels between PEs, and channels between PEs and I/O cards.

Ws2VmePe0 Library

Cores in the Ws2VmePe0 Library are dedicated to designs for PE0 on a WILDSTAR™-II / VME motherboard. It contains cores for onboard static and dynamic memory, channels between PEs, and channels between PEs and I/O cards.

Ws2PciPe0 Library

Cores in the Ws2PciPe0 Library are dedicated to designs for PE0 on a WILDSTAR™-II / PCI motherboard. It contains cores for onboard static and dynamic memory, channels between PEs, and channels between PEs and I/O cards.

FbPciPe0 Library

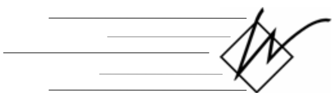
Cores in the FbPciPe0 Library are dedicated to designs for PE0 on a FIREBIRD™ / PCI motherboard. It contains cores for memory, channels between PEs, and channels between PEs and I/O cards.

WsFpdpePe Library

Cores in the WsFpdpePe Library are specific to PE designs on an FPDP-E I/O card residing on a WILDSTAR™ motherboard. Cores in this library provide resources for memory, transmit and receive I/O options, LEDs, and host interrupt.

FbFpdpePe Library

Cores in the FbFpdpePe Library are specific to PE designs on an FPDP-E I/O card residing on a FIREBIRD™ motherboard. Cores in this library provide resources for memory, transmit and receive I/O options, LEDs, and host interrupt.



8. VHDL MODELS REFERENCE

8.1 FPDP-E I/O Card System VHDL Model

This chapter discusses the VHDL Model of the FPDP-E I/O Card as it fits into the WILDSTAR™, WILDSTAR™-II, and WILDSTAR™-II PRO systems. The FPDP-E I/O Card VHDL Model is integrated into the simulation model of the host system. The following sections describe the various user interfaces to the FPDP-E I/O Card system.

Depending on which motherboard you use with the FPDP-E I/O Card, there are some important differences regarding interfaces. Due to motherboard architecture differences, several of the interfaces are board-specific: some are used with WILDSTAR™ and FIREBIRD™ boards, while others are specific to WILDSTAR™-II or WILDSTAR™-II PRO. See Section 8.2.2 for a detailed listing of interface compatibilities.

i

INFORMATION NOTE

I/O daughter cards that have a WILDSTAR™ VME or FIREBIRD™ PCI motherboard can use LAD_Mux components, Mem_Mux components, and the bridges that connect the two. I/O daughter cards that have a WILDSTAR™-II motherboard can only use the Mem_Mux components.

8.1.1 VHDL Model Overview

8.1.1.1 FPDP-E I/O Card Block Diagrams

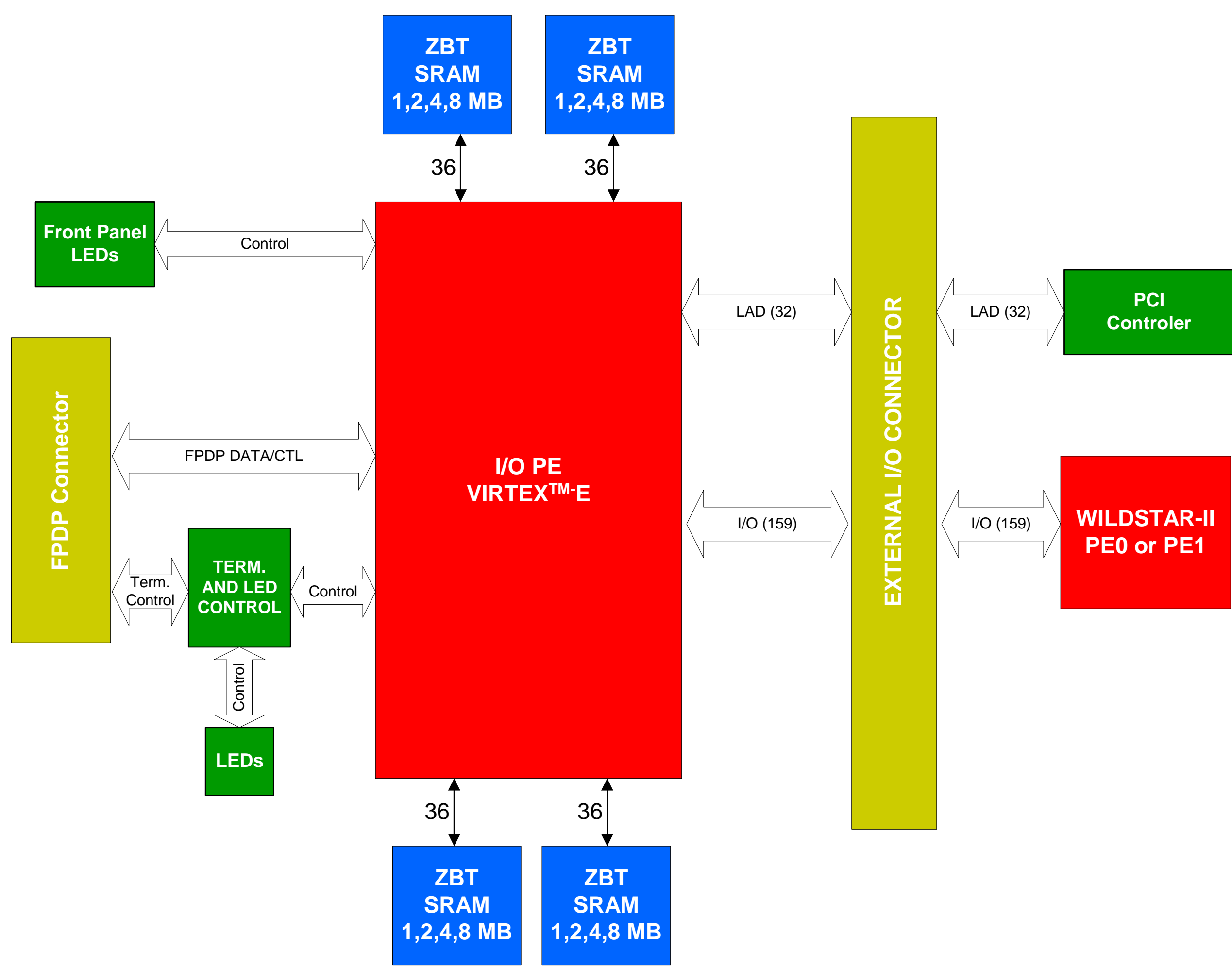


Figure 8-1: FPDP-E I/O Card Block Diagram

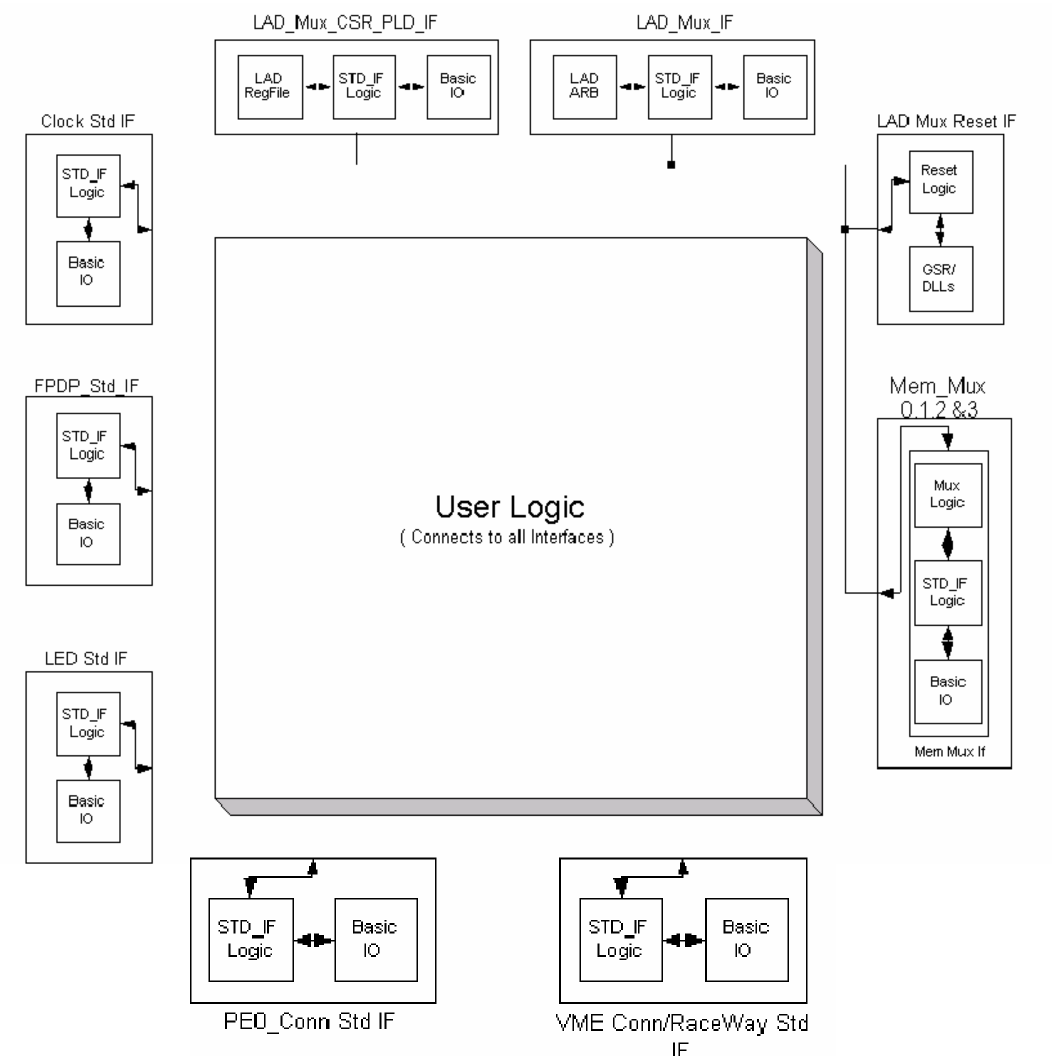


Figure 8-2: FPDP-E I/O Processing Element Block Diagram

8.2 FPDP-E I/O Card PE Model

8.2.1 PE Pad Definitions and Locations

The definitions and locations of the FPDP-E I/O Card PE pads are contained in the pad location data file (fpdpe_io_pe_pad_loc.dat) used by the place-and-route process to constrain the PE pads to the proper locations.

i

INFORMATION NOTE
The pad index corresponds to the logical pad index of the PE VHDL Model, not the physical I/O pin number of the FPGA die.

8.2.2 FPDP-E I/O Card PE Interface Components

Each of the standard interface components are described in the next few sections.

Due to motherboard architecture differences, several of the interfaces will vary depending on the type of motherboard used with the FPDP-E I/O Card. Designs with a WILDSTAR™ and FIREBIRD™ motherboard may use the following interfaces:

- Clock_Std_IF
- LAD_Mux_IF
- LAD_Bus_Std_IF
- LAD_Mux_Reset
- FPDP_Std_IF
- Mem36_Mux_IF
- Mem_Std_IF
- LAD_Mux_CSR_PLD_IF
- LED_Std_IF
- RACEway_Std_IF
- VME_Conn_Std_IF
- PE0_Conn_Std_IF

Designs with a WILDSTAR™-II motherboard may use the following interfaces:

- Clock_Std_IF
- WSII_LAD_Bus_Std_IF
- WSII_Reset_Basic_IO
- FPDP_Std_IF
- Mem36_Mux_IF
- Mem_Std_IF
- CSR_Std_If

- LED_Std_IF
- IO_ConnWS_Basic_IO

Designs with a WILDSTAR™-II PRO motherboard may use the following interfaces:

- Clock_Std_IF
- WSIIPRO_LAD_Bus_Std_IF
- WSII_Reset_Basic_IO
- FPDP_Std_IF
- Mem36_Mux_IF
- Mem_Std_IF
- CSR_Std_If
- LED_Std_IF
- IO_ConnWS_Basic_IO

For a full description of each of the interfaces and components associated with the Mux interfaces, please refer to Section 8.4.

8.2.2.1 Clock Standard Interface (Clock_Std_IF)

The PE Clock_Std_IF is a VHDL component that provides a user interface to the various clock pads and other clock-related information in the VHDL design. The port signals of the Clock_Std_IF are shown in Table 8-1. The "User_In" port signals should be used as the various clock signals in the design.

Each of the clock signals in the clock standard interface component can use the delay locked loop (DLL) feature of the Xilinx® Virtex™ FPGA device. The DLLs help eliminate the effects of clock skew across the FPGA device, but they take time to acquire a lock to the feedback (or reference) clock. The "locked" indicator flags are provided to the design so the user can determine when the DLL has acquired a lock. The clocks may also bypass the DLL component and directly route the clock signal(s) to a global clock buffer (BUFG). This should only be considered when clock skew is not an issue, or a design requires more than four clocks.

The "User_In" port signals of the Clock_Std_IF component should be used as the clock sources of the design. The "Pads" signals of the Clock_Std_IF component are only used to connect the Clock_Std_IF component to the pads.

Table 8-1: Clock Standard Interface Component Port Signals

Signal Name	Width	Dir	Description
Pads.Clocks.M_Clk	1	Input	Corresponds to the M_Clk from the WILDSTAR™ motherboard
Pads.Clocks.K_Clk	1	Input	VME/LAD bus clock pad signal; can be either 33 MHz or 66MHz
Pads.Clocks.U_Clk	1	Input	Corresponds to the U_Clk from the WILDSTAR™ motherboard
Pads.Clocks.P_Clk	1	Input	Corresponds to the P_Clk from the WILDSTAR™ motherboard
Pads.FP_In_Clk	1	Output	FPDP In Clk
Pads.CSR_K_Clk	1	Output	Clock for PLD Shift Register
Pads.U_Clk_USRC	1	Output	Source clk for U_CLK
Pads.U_Clk_MSRC	1	Output	Source clk for M_CLK
Pads.FP_Out_Clk_PECL	1	Output	FPDP Out Clk PECL
Pads.FP_Out_Clk_TTL	1	Output	FPDP Out Clk TTL
Pads.User_In.M_Clk	1	Output	User interface to the Pads.Clocks.M_Clk signal
Pads.User_In.K_Clk	1	Output	User interface to the Pads.Clocks.K_Clk signal
Pads.User_In.U_Clk	1	Output	User interface to the Pads.Clocks.U_Clk signal
Pads.User_In.S_Clk	1	Output	User interface to the Pads.Clocks.S_Clk signal
Pads.User_In.M_Clk_2x	1	Output	M_Clk doubled
Pads.User_In.K_Clk_2x	1	Output	K_Clk doubled
Pads.User_In.U_Clk_2x	1	Output	U_Clk doubled
Pads.User_In.S_Clk_2x	1	Output	S_Clk doubled
User_In.M_Clk_Locked	1	Output	M_Clk DLL lock flag; indicates DLL has achieved a lock when '1'
User_In.K_Clk_Locked	1	Output	K_Clk DLL lock flag; indicates DLL has achieved a lock when '1'
User_In.U_Clk_Locked	1	Output	U_Clk_DLL lock flag; indicates DLL has achieved a lock when '1'
User_In.S_Clk_Locked	1	Output	S_Clk_DLL lock flag; indicates DLL has achieved a lock when '1'
User_Out.M_Clk_DllRst	1	Output	M_Clk Dll Reset
User_Out.K_Clk_DllRst	1	Output	K_Clk Dll Reset
User_Out.U_Clk_DllRst	1	Output	U_Clk Dll Reset
User_Out.S_Clk_DllRst	1	Output	S_Clk Dll Reset
User_Out.FP_Out_Clk_Sel	2	Input	FPDP Out Clk Select
User_Out.Race_Clk	1	Input	Raceway Clk into the clock standard interface
User_Out.Sky_Clk	1	Input	Sky Clk into the clock standard interface

The Clock_Std_IF component also has several generics that can be set by the user upon instantiation of the component:

- **M_CLK_DLL_TYPE**: When set to HIGH_FREQ, the User_In.M_Clk signal is driven by the CLK0 output of a high frequency CLKDLLHF component. When set to LOW_FREQ, the User_In.M_Clk signal is driven by the CLK0 output of a low frequency CLKDLL component. When set to NONE, the User_In.M_Clk signal is driven directly by the clock pad signal (i.e. the CLKDLL component is bypassed). The default setting is LOW_FREQ.

- **U_CLK_DLL_TYPE**: When set to HIGH_FREQ, the User_In.U_Clk signal is driven by the CLK0 output of a high frequency CLKDLLHF component. When set to LOW_FREQ, the User_In.U_Clk signal is driven by the CLK0 output of a low frequency CLKDLL component. When set to NONE, the User_In.U_Clk signal is driven directly by the clock pad signal (i.e., the CLKDLL component is bypassed). The default setting is LOW_FREQ.

- **P_CLK_DLL_TYPE:** When set to HIGH_FREQ, the P_CLK signal is driven by the CLK0 output of a high frequency CLKDLLHF component. When set to LOW_FREQ, the P_CLK signal is driven by the CLK0 output of a low frequency CLKDLL component. When set to NONE, the P_CLK signal is driven directly by the clock pad signal (i.e., the CLKDLL component is bypassed). The default setting is LOW_FREQ. P_CLK can then be selected as the source for User_In.S_Clk. (see S_CLK_SOURCE below)

- **FP_CLK_DLL_TYPE**: When set to HIGH_FREQ, the FP_In_Clk signal is driven by the CLK0 output of a high frequency CLKDLLHF component. When set to LOW_FREQ, the FP_In_Clk signal is driven by the CLK0 output of a low frequency CLKDLL component. When set to NONE, the FP_In_Clk signal is driven directly by the clock pad signal (i.e., the CLKDLL component is bypassed). The default setting is LOW_FREQ. FP_In_Clk can then be selected as the source for User_In.S_Clk. (see S_CLK_SOURCE below)

- **S_CLK_SOURCE**: This generic determines the input to the fourth global clock buffer on the Virtex-E part. The choices are XLDCLK, PCLK, FPCLK, SKYCLK, and NONE.

- **M_CLK_SOURCE**: When set to OSCILLATOR, M_CLK remains sourced by the motherboard. When set to FPCLK, XLDCLK, or SKYCLK, M_CLK will be sourced by the chosen clock.

- **U_CLK_SOURCE:** When set to OSCILLATOR, U_CLK remains sourced by the motherboard. When set to FPCLK, XLDCLK, or SKYCLK, U_CLK will be sourced by the chosen clock.

- **M_CLK_DLL_DIVISOR:** This string generic sets the divisor of the M_Clk DLL. The resultant divided output will be clocked out on the Clocks_In.M_Clk_DV signal. Valid M_CLK_DLL_DIVISOR are "1.5", "2.0", "2.5", "3.0", "3.5", "4.0", "4.5", "5.0", "5.5", "6.0", "6.5", "7.0", "7.5", "8.0", "9.0", "10.0", "11.0", "12.0", "13.0", "14.0", "15.0", and "16.0" The default setting is "2.0".

- **U_CLK_DLL_DIVISOR:** This string generic sets the divisor of the U_Clk DLL. The resultant divided output will be clocked out on the Clocks_In.U_Clk_DV signal. Valid U_CLK_DLL_DIVISOR are "1.5", "2.0", "2.5", "3.0", "3.5", "4.0", "4.5", "5.0", "5.5", "6.0", "6.5", "7.0", "7.5", "8.0", "9.0", "10.0", "11.0", "12.0", "13.0", "14.0", "15.0", and "16.0" The default setting is "2.0".

- **S_CLK_DLL_DIVISOR:** This string generic sets the divisor of the S_Clk DLL. The resultant divided output will be clocked out on the Clocks_In.S_Clk_DV signal. Valid S_CLK_DLL_DIVISOR are "1.5", "2.0", "2.5", "3.0", "3.5", "4.0", "4.5", "5.0", "5.5", "6.0", "6.5", "7.0", "7.5", "8.0", "9.0", "10.0", "11.0", "12.0", "13.0", "14.0", "15.0", and "16.0" The default setting is "2.0".

i

INFORMATION NOTE
Care must be taken not to use more than four global clock buffers in a design. Any unused clock will remain unrouted; therefore, any clock buffer associated with that net will be removed. A design will simulate and synthesize with more than four global clock buffers, but the Place and Route tool will fail.

8.2.2.2 LAD Interfaces

The Local Address/Data Bus is the primary means by which the FPDP-E I/O PE communicates with the host system. There are two different LAD bus specifications, and therefore two different LAD bus interfaces. FPDP-E I/O boards which have a WILDSTAR™ /VME or FIREBIRD™ /PCI mainboard should use the LAD_Mux_IF described in section 8.2.2.2.1 below. FPDP-E I/O boards which have a WILDSTAR™-II mainboard should use the WSII_LAD_Bus_Std_IF described in section 8.2.2.2.2 below.

8.2.2.2.1 LAD Mux Interface (LAD_Mux_IF)

The LAD Mux interface (LAD_Mux_IF) uses the LAD bus path between the PCI Controller and FPDP-E I/O PE device. The LAD bus is a single master (PCI Controller), 32-bit, shared address/data bus. Every cycle on the LAD bus is initiated by the PCI Controller and can last anywhere from four to hundreds of clock (K_Clk) cycles. The signals associated with the LAD_Mux_IF are described in Table 8-2. Furthermore, the LAD Mux Clients record, which is composed of an array of LAD_Mux_Vectors, is described in Table 8-3.

Table 8-2: LAD Mux Interface Component Port Signals

Signal Name	Width	Dir	Description
Kclk	1	Input	Local Address bus clock signal.
Reset	1	Input	Reset (or set) signal; usually connected to the GSR input of a STARUP_VIRTEX component.
Pads.Addr_Data	32	Bi-dir	Local Address Bus shared address/data bus pads signal.
Pads.DS_n	1	Input	Local Address Bus data strobe signal.
Pads.Reg_n	1	Input	Local Address Bus Register access pads signal. This signal is a '0' during register accesses and a '1' during DMA accesses.
Pads.WR_n	1	Input	Local Address Bus Write pads signal.
Pads.DMA_Chan	2	Input	Local Address Bus DMA channel pads signal
Pads.DMA_Stat	2	Output	Local Address Bus DMA status flag pads signal.
Clients	Record	Bi-dir	Array of LAD_Mux_vector records which represent every design unit that accesses the LAD Bus attached to this mux.

Table 8-3: LAD Mux Interface Component Port Signals

Signal Name	Width	Dir	Description
Clients(I).Addr	16	Output	User interface to the LAD bus address; note that this is a DWORD address; also note that this address is automatically incremented each clock cycle during burst LAD bus cycles
Clients(I).Write	1	Output	Write select interface signal; indicates a write cycle when '1' or a read cycle when '0'
Clients(I).Strobe	1	Output	Register access strobe signal; indicates a valid register cycle when '1'
Clients(I).DMA_Strobe	1	Output	DMA access strobe signal; indicates a valid DMA cycle when '1'
Clients(I).Reset	1	Output	Reset generated from the Global_Reset signal.
Clients(I).Data_In	31	Output	User interface to the input data from the LAD Mux. New data from the host is indicated by a low to high transition strobe signal. The data_In register will only change when strobe transitions.
Clients(I).Data_Out	31	Input	User interface to the output data to the LAD bus
Clients(I).Akk	1	input	Acknowledge signal from a LAD Mux client in register space. Only one Akk signal will be high during a single clock cycle.
Clients(I).Int_Req	1	Output	User interface to the interrupt request signal; low-to-high transition on this signal generates a single pulse on the interrupt request pad signal (which in turn will generate a PCI interrupt to the host)
Clients(I).DMA_Chan	2	Output	User interface to the DMA channel number indicator; used to request status of one of four DMA channels in the PEX
Clients(I).DMA_RStat	2	Output	User interface to the DMA read status flags; indicates the status of a DMA read.
Clients(I).DMA_WStat	2	Output	User interface to the DMA read status flags; indicates the status of a DMA write.
Clients(I).DMA_RAkk	1	Output	DMA read acknowledge signal from a LAD Mux client. This signal should always be driven low; the DMA_Read_IF controls this signal.
Clients(I).DMA_WAkk	1	Output	DMA write acknowledge signal from a LAD Mux client. This signal should always be driven low; the DMA_Write_IF controls this signal.

8.2.2.2.2 WILDSTAR™-II LAD Standard Interface (WSII_LAD_Bus_Std_IF)

The LAD Bus is the primary means of communicating with the host system. Using the LAD bus, the PE can send and receive programmed I/O data from the host and transfer data between PEs on a single board.

The component declarations for the LAD_Bus_Std_IF and port record types are shown below:

```
component WSII_LAD_Bus_Std_IF is
  port
  (
    Global_Reset  : in    std_logic;
    Pads          : inout LAD_Bus_Pads_Type;
    User_In       :   out WSII_LAD_Bus_Std_IF_In_Type;
    User_Out      : in    WSII_LAD_Bus_Std_IF_Out_Type
  );
end component;
```

```
type WSII_LAD_Bus_Std_IF_In_Type is record
  Addr            : std_logic_vector ( 18 downto 0 );
  Data_In         : std_logic_vector ( 31 downto 0 );
  Reg_Strobe      : std_logic;
  DMA_Strobe      : std_logic;
  Write           : std_logic;
  PciRdy          : std_logic;
  BusGnt          : std_logic;
  Reset           : std_logic;
end record;
```

```
type WSII_LAD_Bus_Std_IF_Out_Type is record
  Data_Out        : std_logic_vector ( 31 downto 0 );
  Strobe_Out      : std_logic;
  PeRdy           : std_logic;
  IntReq          : std_logic;
  BusReq          : std_logic;
end record;
```

The “User_In” and “User_Out” port signals of the WSII_LAD_Bus_Std_IF component should be used by the design to receive data from and send data to the LAD bus of the FPDP-E I/O device, respectively. The “Pads” signals of the LAD_Bus_Std_IF component are only used to connect the WSII_LAD_Bus_Std_IF component to the pads, and should always be assigned to Pads.LAD_Bus. A definition of each of the port signals is given in Table 8-4 below.

Table 8-4: LAD Bus Standard Interface Component Port Signals

Signal Name	Width	Dir	Description
Pads	NA	Bi-dir	LAD Bus Pad signals
Global_Reset	1	Input	Global reset (or set) signal; usually connected to the GSR input of a STARTUP_Virtex component
User_In.Addr	19	Output	User interface to the LAD bus address; note that this is a DWORD address; also note that this address is automatically incremented each clock cycle during burst LAD bus cycles
User_In.Data_In	32	Output	User interface to the input data from the LAD bus
User_In.Reg_Strobe	1	Output	Register space access strobe signal; indicates a valid register space cycle when '1'
User_In.DMA_Strobe	1	Output	DMA access strobe signal; indicates a valid DMA cycle when '1'
User_In.Write	1	Output	Write select interface signal; indicates a write cycle when '1' or a read cycle when '0' (when strobe is '1')
User_In.PCIRdy	1	Output	Set to '1' when the PCI controller can accept DMA data
User_In.BusGnt	1	Output	Set to '1' when the PE can master the LAD bus
User_In.Reset	1	Output	PE Reset Signal. Can be toggled by a host API call
User_Out.Data_Out	32	Input	User interface to the output data to the LAD bus
User_Out.Strobe_Out	1	Input	The PE drives this signal to '1' during a register read to indicate that the requested data has been driven on the LAD bus.
User_Out.PeRdy	1	Input	'1' indicates that the PE can accept DMA data '0' indicated that the PE cannot accept DMA data
User_Out.IntReq	1	Input	User interface to the interrupt request signal; low-to-high transition on this signal generates a single pulse on the interrupt request pad signal (which in turn will generate a PCI interrupt to the host)
User_Out.BusReq	1	Input	The PE drives this signal to '1' to request LAD bus mastering. The PCI controller will respond by setting User_In.BusGnt to '1' when mastering is possible.

8.2.2.2.2.1 LAD Bus Transactions

There are two types of LAD Bus transaction cycles—register space read cycles and register space write cycles, both of which are initiated by the PCI controller.

If during any given clock cycle the strobe and write select are both active, then a write cycle is taking place and both the address and data are valid during that clock cycle. If during any given clock cycle the strobe is active and the write select is inactive, then a read cycle is taking place and the address is valid during that clock cycle. The PE then has ~64 clock cycles to drive the data for the read onto the outgoing data bus, accompanied by asserting User_Out.Strobe_Out to '1'.

If the PE fails to respond to the read in time, the LAD interface will return an invalid data value and set an error flag that may be read from the host.

8.2.2.2.2.2 Register Space Transactions

A typical LAD bus register write access is illustrated in the timing diagram labeled Figure 8-3:3-633-3. Both the pads and the User_In and User_Out signals are given. On the LAD bus pads, register writes can have bursts or multiple data phases for each address phase. On the user interface signals, however, an address is presented with each data word.

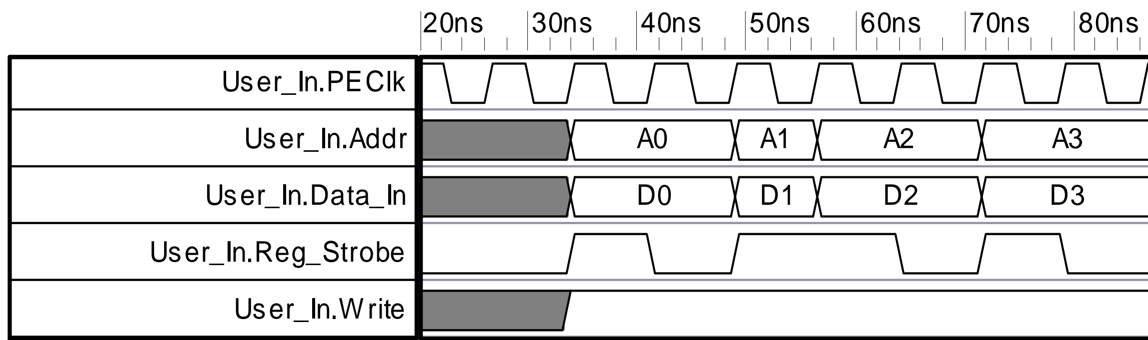


Figure 8-3: Typical LAD Bus Write Cycle from Register Space

Unlike register writes, LAD bus register reads are not bursts. A single data word is expected for every address phase on the LAD bus pads. Within ~64 clock cycles of receiving a register read LAD bus transfer, the PE must simultaneously drive LAD_Bus_Out.Data Out with the requested data and User_Out.Strobe out to '1'.

INFORMATION NOTE
Although 64 clock cycles are allowed, long delays will halt all PCI bus traffic and degrade system performance. Delays should be kept as short as possible.

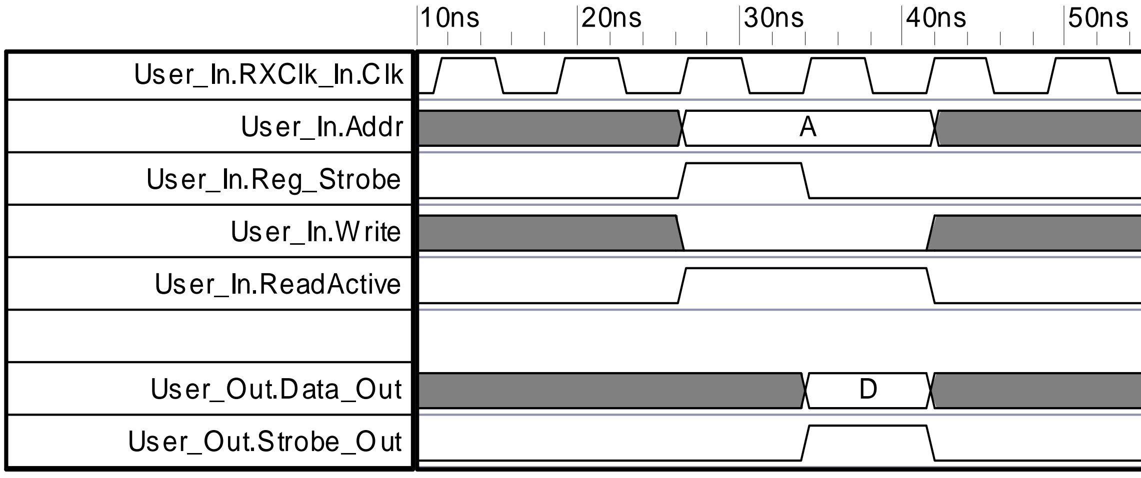


Figure 8-4: Shortest Possible Register Space LAD Bus Read Cycle

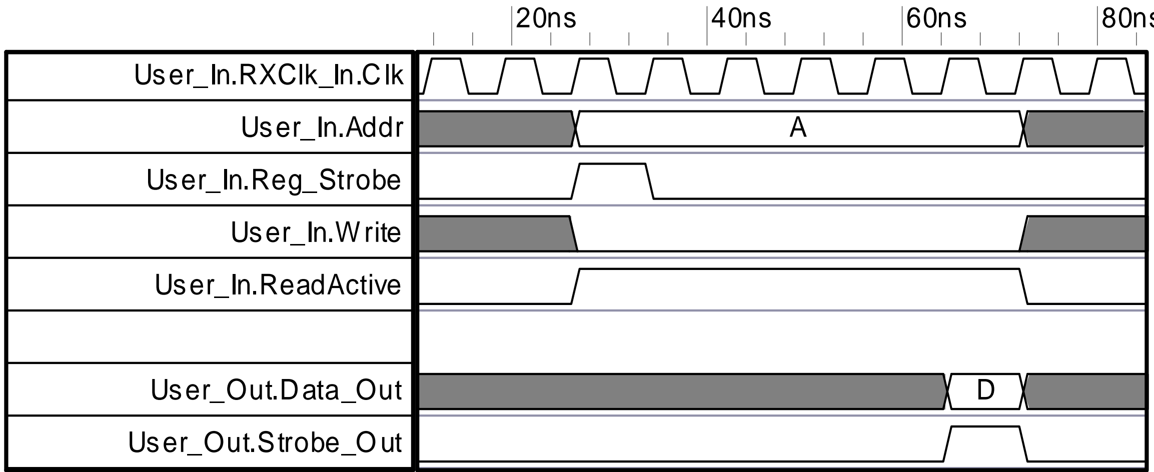


Figure 8-5: Longer Register Space LAD Bus Read Cycle

8.2.2.2.3 WILDSTAR™-II PRO LAD Standard Interface (WSIIPRO_LAD_Bus_Std_IF)

This WILDSTAR™-II PRO LAD Bus standard interface interfaces to WILDSTAR™-II PRO mainboard LAD bus. There are two reserved registers built into the interface itself.

The component declarations for the WSIIPRO_LAD_Bus_Std_IF and port record types are shown below:

```
component WSIIPRO_LAD_Bus_Std_IF is
  port
  (
    Clk             : in    std_logic;
    Global_Reset    : in    std_logic;
    PadsVME         : inout VME_Conn_Pads_Type;
    Pads            : inout LAD_Bus_Pads_Type;
    User_In         :   out WSII_LAD_Bus_Std_IF_In_Type;
    User_Out        : in    WSII_LAD_Bus_Std_IF_Out_Type
  );
end component;
```

```
  type WSII_LAD_Bus_Std_IF_In_Type is record
    Addr              : std_logic_vector ( 18 downto 0 );
    Data_In           : std_logic_vector ( 31 downto 0 );
    Reg_Strobe        : std_logic;
    DMA_Strobe        : std_logic;
    Write             : std_logic;
    PciRdy            : std_logic;
    BusGnt            : std_logic;
    Reset             : std_logic;
  end record;
```

```
type WSII_LAD_Bus_Std_IF_Out_Type is record
  Data_Out        : std_logic_vector ( 31 downto 0 );
  Strobe_Out      : std_logic;
  PeRdy           : std_logic;
  IntReq          : std_logic;
  BusReq          : std_logic;
end record;
```

The WILDSTAR™-II PRO LAD Bus standard interface uses the same ports that the WILDSTAR™-II non-PRO LAD Bus standard interface uses.

8.2.2.3 WILDSTAR™-II PRO LAD Bus Standard Interface Reserved Registers

The WILDSTAR™-II PRO LAD Bus has two built-in registers. These two registers are a version register and a signature register. Both of these registers are read-only and contain only static information.

INFORMATION NOTE
An entire block of address space is blocked off for internal use with this interface. This space goes from 0x00 to 0x3C on each PE. Registers should not be created in these locations.

Version Register (0x0): This register contains the VHDL version. Bits 23 to 16 contain the major release number. Bits 15 to 8 contain the minor release number. Bits 7 to 0 contain the second minor release number.

Signature Register (0x3C): This register is reserved by the software as a test register only.

8.2.2.4 LAD_Mux_Reset

The component instantiation for the LAD_Mux_Reset is shown in the following code listing:

```
u_LAD_Mux_Reset: LAD_Mux_Reset
  generic map
  (
    Base        => x"7ff8"
  )
  port map
  (
    Rclk        => M_Clk,
    Kclk        => K_Clk,
    LAD         => LAD_Bus(4),

    Reset       => Global_Reset,
    DLL_Reset_0 => Clocks_Out.M_Clk_DllRst,
    DLL_Reset_1 => Clocks_Out.P_Clk_DllRst,
    DLL_Reset_2 => Clocks_Out.K_Clk_DllRst,
```

```
        DLL_Reset_3 => Clocks_Out.U_Clk_DllRst
    );
```

The LAD_Mux_Reset unit provides an LAD accessible reset unit for the PE. It encapsulates a VIRTEX_STARTUP block. When a '1' is written to the unit's single control address, it will generate a reset pulse on the global reset line that lasts for several Kclk periods. The reset pulse is also available at the Reset port for use in simulation.

In addition, the reset unit provides a standard method for resetting the on-chip clock DLLs. While Bit 0 of the unit's control register provides the global reset pulse, bits 1-4 provide control over the four DLLs. Writing *0x1fe* to the reset unit will place all four DLLs in reset. Writing *0x0* to the reset unit will remove the reset applied to the DLLs, and bring them back out of reset. Although DLL_Reset ports are intended for DLLs, they can be used as additional resets for any logic inside the PE's design.

The LAD_Mux_Reset unit uses a single address. In the case above, the LAD_Mux_Reset unit responds only to LAD bus address 0x7ff8.

INFORMATION NOTE
In the above instantiation, the LAD_Mux_Reset unit has been connected to the fourth element of the LAD_Bus signal. As stated above, this element must not be shared with another client.

8.2.2.5 LAD Mux CSR PLD Interface (LAD_Mux_CSR_PLD_IF)

INFORMATION NOTE
The **LAD_Mux_CSR_PLD_IF** encapsulates and replaces the **CSR_PLD_Std_IF.** Please refer to Attachment C for more information on the **CSR_PLD_Std_IF**.

The Control and Status Register (CSR) PLD Mux Interface is a VHDL component that provides a user interface to the CSR PLD from within the FPDP-E PE device.

The LAD_Mux_CSR_PLD_IF incorporates the CSR_PLD_Std_IF, so both interfaces operate in a similar manner. However, the LAD_Mux_CSR_PLD_IF allows the user to connect a LAD_Mux_vector directly to the CSR PLD interface, for easy communication between the host or the PE and the CSR PLD. The following table describes the port signals of the LAD_Mux_CSR_PLD_IF component.

Table 8-4: LAD Mux CSR PLD Interface Component Port Signals

Signal Name	Width	Dir	Description
Kclk	31	Input	Local Address bus clock signal.
Global_Reset	1	Input	Reset (or set) signal; usually connected to the GSR input of a STARUP_VIRTEX component.
Mode 1	36	Input	Mode1 signal
Mode 2	36	Input	Mode2 signal
Method	1	Input	Termination Method signal
Pads.Data	1	Bi-Dir	Bi-directional PLD data pad signal
Pads.Strobe_n	1	Output	PLD strobe pad signal
Pads.WR_n	1	Output	PLD read/write pad signal
LAD	Record	Bi-Dir	LAD_Mux_Vector record

Table 8-5 describes the LAD_MUX_Vector inputs that can be used to control the CSR PLD.

Table 8-5: LAD Mux CSR PLD Interface Component Port Signals

Register Bit	Description
.Data_In(0 to 2)	User interface to the LEDs.
Data_In(3)	User interface to the PIO2 direction bit.
Data_In(4)	User interface to the PIO1 direction bit.
.Data_In(5)	User interface to the CLK In Select (0 PECL / 1 TTL) (N/A if FPDP TM)
.Data_In(6)	User interface to the TTL Clock Enable (0 TTL / 1 PECL)
.Data_In(13 to 7)	Reserved
.Data_Out(0 to 6)	User interface to the values stored in the PLD (Same as Data_In(0 to 6))
.Data_Out(7 to 10)	User interface to the status of the power sleep mode for memories 0 to 3
.Data_Out(11)	User interface to the MCLK error bit
.Data_Out(12)	User interface to the UCLK Master bit
.Data_Out(13)	User interface to the status of the 1.8 Power supply.
.Data_Out(31)	When high indicates the PLD has been successfully programmed.

8.2.2.6 Mem36_Mux_IF (Priority and Fair)

The FPDP-E card has access to four identical 36-bit on-board synchronous SRAM memory ports that are referred to as Mem_Mux0, Mem_Mux1, Mem_Mux2, and Mem_Mux3. The Mem36 mux signals are described in the table below. For more information on the Mem36_Mux_IF please refer to section 8.4.3 and Attachment A.

Table 8-6: Mem 36 Mux Vector Signals

Signal Name	Width	Dir	Description
Clients(I).Addr	31	Output	Memory Address bus signals
Clients(I).Write	1	Input	Memory Write Select (1 = Write, 0 = Read)
Clients(I).Data_In	36	input	Data from memory
Clients(I).Data_Out	36	Output	Data to memory
Clients(I).Data_Valid	1	Output	Signal which represents valid data from a memory read.
Clients(I).Req	1	Output	Indicates that the operation being presented by the client is valid. Req must be asserted by the client during a memory access
Clients(I).Akk	1	Input	Indicates that the memory interface has accepted the transaction and will attempt to complete it

8.2.2.7 FPDP Standard Interface (FPDP_Std_IF)

The FPDP standard interface provides access to the FPDP. The port signals of the FPDP standard interface are described in Table 8-7. The “User_In” and “User_Out” port signals should be used in the design to receive data from and send data to the FPDP port, respectively. This interface registers inputs and outputs in the IOBs.

Table 8-7: FPDP Standard Interface Component Port Signals

Signal Name	Width	Dir	Description
Pads.FPDP.D	32	Bi-dir	FPDP Data Bit
Pads.FPDP.DIR_n	1	Bi-dir	FPDP Data Direction
Pads.FPDP.DVALID_n	1	Bi-dir	FPDP Data Valid
Pads.FPDP.NRDY_n	1	Bi-dir	FPDP Not Ready
Pads.FPDP.PIO1	1	Bi-dir	FPDP Programmable I/O
Pads.FPDP.PIO2	1	Bi-dir	FPDP Programmable I/O
Pads.FPDP.SUSPEND_n	1	Bi-dir	FPDP Suspend Data
Pads.FPDP.SYNC_n	1	Bi-dir	FPDP Sync Pulse
Global_Reset	1	Input	Asynchronous reset signal for IOB registers
Clk	1	Input	Clock signal for IOB registers
User_In.D_In	32	Output	In receive mode, incoming data arrives on this port
User_In.DIR_In_n	1	Output	In receive mode, a low signal indicates that a transmit master is present on the bus
User_In.DVALID_In_n	1	Output	In receive mode, a low signal indicates that valid data is available on the incoming data port
User_In.NRDY_In_n	1	Output	In transmit mode, a low signal indicates that the receiver is not ready to receive data
User_In.PIO1_In	1	Output	When PIO1 is configured as an input, incoming signals arrive on this port
User_In.PIO2_In	1	Output	When PIO1 is configured as an input, incoming signals arrive on this port
User_In.SUSPEND_In_n	1	Output	signaling a pending buffer overflow condition
User_In.SYNC_In_n	1	Output	In receive mode, the transmitter provides synchronization using this signal
User_Out.D_Out	32	Input	In transmit mode, the transmitter drives data across the FPDP bus using this port
User_Out.D_OE_n	32	Input	Output enables for the User_Out.D_Out port
User_Out.DIR_Out_n	1	Input	In transmit mode, the transmitter drives this signal low
User_Out.DIR_OE_n	1	Input	Output enable for the User_Out.Dir_Out_n port
User_Out.DVALID_Out_n	1	Input	In transmit mode, the transmitter drives this signal low when it is presenting valid data at the output port
User_Out.DVALID_OE_n	1	Input	Output enable for the ser_Out.DVALID_Out_n port
User_Out.NRDY_Out_n	1	Input	In receive mode, the receiver can indicate that it is not ready to receive data by driving this signal low
User_Out.NRDY_OE_n	1	Input	Output enable for the User_Out.NRDY_Out_n port
User_Out.PIO1_Out	1	Input	User programmable I/O pin
User_Out.PIO2_Out	1	Input	User programmable I/O pin
User_Out.PIO1_OE_n	1	Input	User programmable I/O pin Output Enable
User_Out.PIO2_OE_n	1	Input	User programmable I/O pin Output Enable
User_Out.SUSPEND_Out_n	1	Input	In receive mode, the receiver may drive this

Signal Name	Width	Dir	Description
			signal low to indicate a pending buffer overflow condition
User_Out.SUSPEND_OE_n	1	Input	Output enable for the User_Out.SUSPEND_Out_n port
User_Out.SYNC_Out_n	1	Input	In transmit mode, the transmitter may use this signal to provide synchronization
User_Out.SYNC_OE_n	1	Input	Output enable for the User_Out.SYNC_OE_n port

8.2.2.8 LED Standard Interface (LED_Std_IF)

The LED_Std_IF provides the user interface to the light emitting diodes (LED) of the FPDP-E I/O card PE device. The LED can be used to indicate status or processing activity, and is useful in debugging a PE design. The port signals of the LED standard interface are shown in Table 8-8. The active low "User_Out" signals should be used to drive the LED of the PE device.

Table 8-8: LED Standard Interface Component Port Signals

Signal Name	Width	Dir	Description
Pads	2	Output	LED pad signal
User_Out_n	2	Input	User interface to the front panel LEDs; signals are active low

8.2.2.9 RACEway VME Backplane Connector Standard Interface (RACEway_Std_IF)

INFORMATION NOTE

The FPDP-E I/O Card is a RACEway™-ready product. Annapolis Micro Systems, Inc. sells an FPGA core that will run with the FPDP-E I/O Card FPGA. The FPDP-E I/O Card VHDL contains an optional RACEway pin interface to allow the user to design a custom RACEway core.

The RACEway_Std_IF component provides the user interface to the backplane RACEway signals. The WILDSTAR™ I/O connector is 39 bits wide. The port signals of the RACEway_Std_IF component are described in Table 8-9. The "User_In" and "User_Out" port signals should be used in the design to receive data from and send data to the I/O connectors, respectively.

Table 8-9: RACEway™ Connector Standard Interface Component Port Signals

Signal Name	Width	Dir	Description
Global_Reset	1	Input	Asynchronous reset signal for IOB registers
Clk	1	Input	Clock signal for IOB registers
User_In.Data_In	32	Input	RACEway Data In
User_In.Data_Out	32	Output	RACEway Data Out
User_In.Sync_n	1	Input	RACEway Control Phase 1
User_In.Reset_In_n	1	Input	RACEway Reset In
User_In.Reset_Out_n	1	Output	RACEway Reset Out
User_In.Req_In	1	Input	RACEway Kill
User_In.Reply_In	1	Input	RACEway Change to Address, Data Strobe Enable, Read Ready, and Split In
User_In.Reply_Out	1	Output	RACEway Change to Address, Data Strobe Enable, Read Ready, and Split Out
User_In.Reply_OE	1	Output	RACEway Change to Address, Data Strobe Enable, Read Ready, and Split Output Enable
User_In.Strobio_In	1	Input	RACEway Astrobe, Data Strobe In
User_In.Strobio_Out	1	Output	RACEway Astrobe, Data Strobe Out
User_In.Strobio_OE_n	1	Output	RACEway Astrobe, Data Strobe Output Enable
User_In.Rdconio_In	1	Input	RACEway RDCON Input
User_In.Rdconio_Out	1	Output	RACEway RDCON Output
User_In.Rdconio_OE_n	1	Output	RACEway RDCON Output Enable
User_In.Race_Clk	1	Input	RACEway Clk to be routed to clock std interface if used

8.2.2.10 Motherboard I/O Connectors

The FPDP-E I/O Card has a number of data bits connecting the I/O PE to the motherboard.

When using a WILDSTAR™ /VME motherboard, these bits are divided across two interfaces, VME_Conn_Std_IF and PE0_Conn_Std_IF. The VME_Conn_Std_If connects 96 bits from the FPDP-E I/O PE to the VME backplane. If the FPDP-E I/O PE is in slot 0 of the motherboard, these bits connect to the P0 backplane; otherwise the bits are connected to the P2 backplane. The PE0_Conn_Std_IF connects 66 bits in the FPDP-E I/O PE to a WILDSTAR™ motherboard PE0.

When using a FIREBIRD™ motherboard both the VME_Conn_Std_IF and the PE0_Conn_Std_IF data bits connect directly to the FIREBIRD™ motherboard PE.

When using a WILDSTAR™-II motherboard, the WSII_IOConn_Basic_IO must be used. This interface divided the connector bits into 153 data bits and six clock control lines.

When using a WILDSTAR™-II PRO motherboard, the IOConn_Type1_Basic_IO must be used. This interface divides the connector bits into 152 data bits and six clock control lines.

8.2.2.10.1 VME Backplane Connector Standard Interface (VME_Conn_Std_IF)

The VME_Conn_Std_IF component provides the user interface to the bi-directional connections to external I/O devices. The WILDSTAR™ I/O connector is 96 bits wide and is presented to the user as an asynchronous input and output port (with enable). The port signals of the VME_Conn_Std_IF component are described in Table 8-10. The "User_In" and "User_Out" port signals should be used in the design to receive data from and send data to the I/O connectors, respectively.

Table 8-10: VME Connector Standard Interface Component Port Signals

Signal Name	Width	Dir	Description
Pads	96	Bi-dir	I/O connector pad signals
User_In.Data_In	96	Output	User interface to the VME I/O connector data input
User_Out.Data_Out	96	Input	User interface to the VME I/O connector data output
User_Out.Data_OE_n	96	Input	User interface to the VME I/O connector data output enables; each data bit driver is enabled when its respective output enable is '0' or disabled (for reading) when '1'

Table 8-11: VME Connector Standard Interface Motherboard Signal Mapping

Motherboard	Motherboard Interface	Motherboard Data Bits	VME_Conn_Std_IF Data Bits
FIREBIRD™	IO_Conn_Std_IF	[66:161]	[0:96]
WILDSTAR™ /VME	NA	NA	NA
WILDSTAR™-II	NA	NA	NA

8.2.2.10.2 PE0 Connector Standard Interface (PE0_Conn_Std_IF)

The PE0_Conn_Std_IF component provides the user interface to the bi-directional connections to PE0 on the WILDSTAR™/VME motherboard. The PE0 I/O connector is 66 bits wide and is presented to the user as an asynchronous input and output port (with enable). The port signals of the PE0_Conn_Std_IF component are described in the table below. The "User_In" and "User_Out" port signals should be used in the design to receive data from and send data to the I/O connectors, respectively.

Table 8-12: PE0 Connector Standard Interface Component Port Signals

Signal Name	Width	Dir	Description
Pads	66	Bi-dir	I/O connector pad signals
User_In.Data_In	66	Output	User interface to the PE0 I/O connector data input
User_Out.Data_Out	66	Input	User interface to the PE0 I/O connector data output
User_Out.Data_OE_n	66	Input	User interface to the PE0 I/O connector data output enables; each data bit driver is enabled when its respective output enable is ‘0’ or disabled (for reading) when ‘1’

Table 8-13: PE0 Connector Standard Interface Motherboard Signal Mapping

Motherboard	Motherboard Interface	Motherboard Data Bits	PE0_Conn_Std_IF Data Bits
FIREBIRD™	IO_Conn_Std_IF	[0:65]	[0:65]
WILDSTAR™ /VME	IO_Conn_Std_IF	[0:65]	[0:65]
WILDSTAR™-II	NA	NA	NA

8.2.2.10.3 WILDSTAR™-II I/O Connector Basic Interface (IO_ConnWS_Basic_IO)

The IO_ConnWS_Basic_IO provides the user interface to the connections to the motherboard PE on the WILDSTAR™-II motherboard. This interface divides the connector bits into 153 data bits and six clock control lines. These signals are described in the table below.

Table 8-14: WILDSTAR™-II I/O Connector Component Port Signals

Signal Name	Width	Dir	Description
Pads	NA	Bi-dir	I/O connector pad signals
User_In.Data_In	153	Output	User interface to the motherboard PE I/O connector data input
User_In.DLL_Reset_In	1	Output	Dedicated Input Line: Suggested use as a DLL reset signal.
User_In.DLL_Locked_In	1	Output	Dedicated Input Line: Suggested use as a DLL locked signal.
User_In.RxClk	1	Output	Dedicated Input Line: Suggested use as the receive clock for the User_In.Data_In Lines.
User_Out.Data_Out	153	Input	User interface to the motherboard PE I/O connector data output
User_Out.Data_OE_n	153	Input	User interface to the motherboard PE I/O connector data output enables; Each data bit driver is enabled when its respective output enable is ‘0’ or disabled (for reading) when ‘1’
User_Out.DLL_Reset_Out	1	Input	Dedicated Output Line: Suggested use as a DLL reset signal.
User_Out.DLL_Locked_Out	1	Input	Dedicated Output Line: Suggested use as a DLL locked signal.
User_Out.TxClk	1	Input	Dedicated Output Line: Suggested use as the transmit clock for the User_Out.Data_Out Lines.

Table 8-15: WILDSTAR™-II I/O Connector Motherboard Signal Mapping

Motherboard	Motherboard Interface	Motherboard Port Signal	IO_ConnWS_Basic_ IO Port Signal
FIREBIRD™	NA	NA	NA
WILDSTAR™ /VME	NA	NA	NA
WILDSTAR™-II	IO_ConnWS_Basic_IO	Data [0:153]	Data [0:153]
		DLL_Locked_Out	DLL_Locked_In
		DLL_Locked_In	DLL_Locked_Out
		DLL_Reset_Out	DLL_Reset_In
		DLL_Reset_In	DLL_Reset_Out
		TxClk	RxClk
		RxClk	TxClk

8.2.2.10.4 WILDSTAR™-II PRO I/O Connector Basic Interface (IOConn_Type1_Basic_IO)

The IOConn_Type1_Basic_IO provides the user interface to the connections to the motherboard PE on the WILDSTAR™-II PRO motherboard. This interface divides the connector bits into 152 data bits and six clock control lines. These signals are described in the table below.

Table 8-16: WILDSTAR™-II PRO I/O Connector Component Port Signals

Signal Name	Width	Dir	Description
Pads	NA	Bi-dir	I/O connector pad signals
User_In.data_in	152	Output	User interface to the motherboard PE I/O connector data input
User_In.rx_reset_in	1	Output	Dedicated Input Line: Suggested use as a DLL reset signal.
User_In.tx_locked_in	1	Output	Dedicated Input Line: Suggested use as a DLL locked signal.
User_Out.data_out	152	Input	User interface to the motherboard PE I/O connector data output
User_Out.data_oe_n	152	Input	User interface to the motherboard PE I/O connector data output enables; Each data bit driver is enabled when its respective output enable is '0' or disabled (for reading) when '1'
User_Out.tx_reset_out	1	Input	Dedicated Output Line: Suggested use as a DLL reset signal.
User_Out.rx_locked_out	1	Input	Dedicated Output Line: Suggested use as a DLL locked signal.
User_Out.tx_clock_out	1	Input	Dedicated Output Line: Suggested use as the transmit clock for the User_Out.Data_Out Lines.

8.3 FPDP Cables

Each FPDP-E I/O card has an 80-pin FPDP connector that can be connected to another FPDP I/O card or any other board which has an FPDP connector. In order to simulate a FPDP cable, the FPDP-E VHDL model includes a cables package.

The fpdp_cables_package.vhd, located in the

"annapolis/io_card/fpdpe_io_card/vhdl/connectors" directory, contains the component
declarations for each of the component.

As stated above, the FPDP-E cables can be used to connect two or more FPDP-E I/O cards together during simulation. The cables must be driven by a single FPDP_Xmt_Cable, described in section 8.3.1.1, but can be connected to multiple FPDP_Rcv_Cables, described in 8.3.1.2. The signals of the FPDP-E cable are described in the table below.

Table 8-17: FPDP-E Cable Signals

Signal Name	Width	Dir	Description
FPDP_Strobe	1	Bi	Dir*
FPDP_PStrobe	1	Bi	Dir*
FPDP_D	32	Bi	Dir*
FPDP_DIR_n	1	Bi	Dir*
FPDP_DVALID_n	1	Bi	Dir*
FPDP_NRDY_n	1	Bi	Dir**
FPDP_PIO1	1	Bi	Dir***
FPDP_PIO2	1	Bi	Dir***
FPDP_SUSPEND_n	1	Bi	Dir**
FPDP_SYNC_n	1	Bi	Dir*
FPDP_Strobe	1	Bi	Dir*

* Transmit Master drives this signal.
**Receive master drives this signal
***T/M or R/M can drive this signal, depending on the CSR PLD status.

8.3.1 FPDP-E Cable Components

There are two components in the fpdp_cables_pkg.vhd file, the FPDP_Xmit_Cable component and the FPDP_Rev_Cable component. Both components are described below.

8.3.1.1 FPDP Transmit Cable Component

The component declarations for the FPDP-E Transmit cable component are listed below.

```
component FPDP_Xmit_Cable
 port
 (
 Strobe : in std_logic;
 PStrobe : in std_logic;
 D : in std_logic_vector(31 downto 0);
 DIR_n : inout std_logic;
 DVALID_n : inout std_logic;
 SYNC_n : inout std_logic;
 PIO1 : inout std_logic;
 PIO2 : inout std_logic;
 NRDY_n : inout std_logic;
 SUSPEND_n : inout std_logic;
 Cabl : inout FPDP_cable
 );
end component;
```

The FPDP_Xmit_Cable component is used to transmit the clock and data to a FPDP_Rcv_Cable component. The clock signals are unidirectional; however, all other signals are bi-directional. Each of the signals in the component, with the exception of the "cabl" signal, connects directly to the signals available in the FPDP template. The "cabl" signal is connected to one of seven predefined FPDP_Cable components in the fpdp_cables_package.vhd file. (If more than seven cables are required, simply modify the number of cables available.)

8.3.1.2 FPDP Receive Cable Component

The component declarations for the FPDP-E Receive cable component are listed below.

```
component FPDP_Rcv_Cable
port
 (
 Strobe : out std_logic;
 PStrobe : out std_logic;
 D : out std_logic_vector(31 downto 0);
 DIR_n : inout std_logic;
 DVALID_n : inout std_logic;
 SYNC_n : inout std_logic;
 PIO1 : inout std_logic;
 PIO2 : inout std_logic;
 NRDY_n : inout std_logic;
 SUSPEND_n : inout std_logic;
 Cabl : inout FPDP_cable
 );
end component;
```

The FPDP_Rcv_Cable component is used to receive the clock and data from aFPDP_Xmit_Cable component. The clock signals are unidirectional; however, all other signals are bi-directional. Each of the signals in the component, with the exception of the "cabl" signal, connects directly to the signals which are available in the FPDP template. The "cabl" signal is connected to one of seven predefined FPDP_Cable components in the fpdp_cables_package.vhd file. (If more than seven cables are required, simply modify the number of cables available.)

8.4 The WILDSTAR™ Family Mux Library

The WILDSTAR™ Family Standard Library, which encompasses STARFIRE™, is a set of components and interface specifications that simplify application development by providing commonly used interfaces and promoting reusable components. The current release of the WILDSTAR™ Family Mux Library contains two major component groups—the LAD_Mux library and the Mem_Mux library. The LAD_Mux library simplifies the most common LAD Bus module development. The Mem_Mux library abstracts the interface to the available memory ports on the WILDSTAR™ board and provides the capability to share those interfaces between PEs.

INFORMATION NOTE

I/O daughter cards that have a WILDSTAR™ VME or FIREBIRD™ PCI motherboard can use LAD_Mux components, Mem_Mux components, and the bridges that connect the two. I/O daughter cards that have a WILDSTAR™-II motherboard can only use the Mem_Mux components.

Two types of components are provided by the libraries—server and client. A "server" is a component that allows multiple modules to share a single resource. Servers are characterized by the rules used to arbitrate between modules competing for access to the resource. A "client" is a component that requires access to a resource controlled by a server. A client must obey certain protocols when negotiating with a server for access. Figure 8-6655 illustrates the Mux Library Scheme.

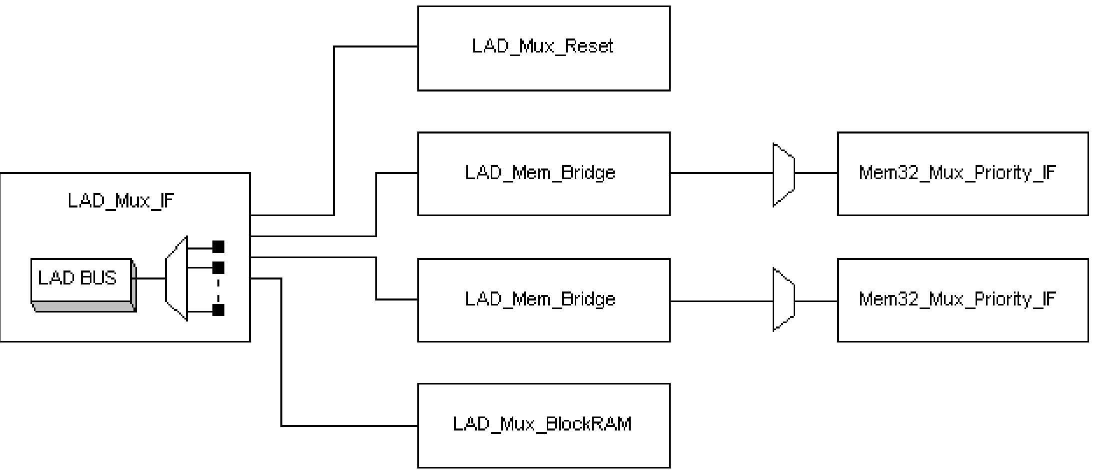


Figure 8-6: General Mux Library Scheme

8.4.1 Standard PE Templates

The WILDSTAR™ Family Library components that replace the Interface Standard components are provided in the standard template files for each of the WILDSTAR™ PEs. The LAD_Std_If is replaced by the LAD_Mux_If. Each of the Memory standard interfaces is replaced by a Memory Mux interfaces. The template files also include the LAD_Mux_Reset component as a replacement for the standard global reset logic.

8.4.2 LAD_Mux Library

The LAD_Mux Library provides a reusable set of components that allows the user to create fully functional LAD bus interfaces with minimum VHDL coding. The library consists of an LAD bus server unit and a set of possible client components that connect to an arbiter. The LAD bus server is used in place of the LAD_Std_IF. The arbiter has a user-defined number of connections for client components. To construct an LAD interface, the user simply connects appropriate clients to the server.

Clients of the LAD_Mux have a port called LAD. The definition of a LAD_Mux is shown in Table 8-18:

Table 8-18: LAD_Mux Record Signals

Signal Name	Width	Dir	Description
Addr	23	Output	User interface to the LAD bus address; note that this is a QWORD address; also note that this address is automatically incremented each clock cycle during burst LAD bus cycles
Write	1	Output	Write select signal; indicates a write cycle when ‘1’ or read cycle when ‘0’
Strobe	1	Output	Data strobe signal; indicates write data is valid on LAD bus
DMA_Strobe	1	Output	DMA Data strobe signal; indicates write DMA data is valid on LAD bus
Reset	1	Input	User interface to the reset (or set) signal; usually connected to the Global_Reset signal of a LAD_MUX_Reset component.
Data_In	32	Output	User interface to the input data from the LAD bus
Data_Out	32	Input	User interface to the output data to the LAD bus
Akk	1	Input	User interface to the LAD acknowledge signal. Indicates which LAD_Mux_Vector has access to the LAD bus. Only one AKK should be valid at a time.
Int_Req	1	Input	User interface to the interrupt request signal; high-to-low transition on this signal generates a single pulse on the interrupt request pad signal (which in turn will generate a PCI interrupt to the host)
DMA_Rstat	2	Input	User interface to the DMA Read status flags; indicates the read status of a DMA operation.
DMA_WStat	2	Input	User interface to the DMA Write status flags; indicates the write status of a DMA operation.
DMA_RAkk	1	Input	User interface to the DMA Read acknowledge signal. Indicates which LAD_Mux_Vector Read status flags are valid. Only one DMA_RAkk should be valid at a time.
DMA_Wakk	1	Input	User interface to the DMA Read acknowledge signal. Indicates which LAD_Mux_Vector Write status flags are valid. Only one DMA_RAkk should be valid at a time.

Clients are connected to the LAD_Mux server by assigning them an unused element of the LAD_Bus vector. Clients cannot share elements of the Clients vector—each must have its own.

INFORMATION NOTE

All available elements of a LAD_Mux_vector must be connected to a client. Unconnected elements will cause mapping failures in Synplify®.

8.4.2.1 LAD_Mux_IF

The component instantiation for the LAD_Mux_IF is shown in the following code listing:

```
U_LAD_MUX_IF : LAD_Mux_IF
  generic map
  (
    K_Clk_x2 => false
  )
  port map
  (
    Kclk    => K_Clk,
    Reset   => Global_Reset,
    pads    => Pads.LAD_Bus,
    Clients => LAD_Bus
  );
```

The Clients port is bound to an array of LAD_Mux_vector records. Each record corresponds to a single LAD Mux client, such as a Register File, Block RAM, or Memory Bridge. The size of the Clients array must equal the number of clients connected to the LAD bus. The KClk, Reset, and Pads ports should be connected to the appropriate resources. There is also a generic that can be instantiated by the user:

- **K_Clk_x2**: This Boolean generic is used to switch the LAD_Mux interface between 33 MHz LAD bus mode and 66 MHz LAD bus mode. When the K_Clk_x2 generic is set to true, extra register delays are added to provide correct timing for 66 MHz operation

INFORMATION NOTE

All available elements of a LAD_Mux_vector must be connected to a client. Unconnected elements will cause mapping failures in Synplify®.

8.4.2.2 LAD_Mux_Reset

The component instantiation for the LAD_Mux_Reset is shown in the following code listing:

```
u_LAD_Mux_Reset: LAD_Mux_Reset
  generic map
  (
    Base        => x"7ff8"
  )
  port map
  (
    Rclk        => M_Clk,
    Kclk        => K_Clk,
    LAD         => LAD_Bus(4),

    Reset       => Global_Reset,
    DLL_Reset_0 => Clocks_Out.M_Clk_DllRst,
    DLL_Reset_1 => Clocks_Out.P_Clk_DllRst,
    DLL_Reset_2 => Clocks_Out.K_Clk_DllRst,
    DLL_Reset_3 => Clocks_Out.U_Clk_DllRst
 );
```

The LAD_Mux_Reset unit provides an LAD accessible reset unit for the PE. It encapsulates a VIRTEX_STARTUP block. When a ‘1’ is written to the unit's single control address, it will generate a reset pulse on the global reset line that lasts for several Kclk periods. The reset pulse is also available at the Reset port for use in simulation.

In addition, the reset unit provides a standard method for resetting the on-chip clock DLLs. While Bit 0 of the unit's control register provides the global reset pulse, bits 1-4 provide control over the four DLLs. Writing *0x1fe* to the reset unit will place all four DLLs in reset. Writing *0x0* to the reset unit will remove the reset applied to the DLLs, and bring them back out of reset. Although DLL_Reset ports are intended for DLLs, they can be used as additional resets for any logic inside the PE’s design.

The LAD_Mux_Reset unit uses a single address. In the case above, the LAD_Mux_Reset unit responds only to LAD bus address 0x7ff8.

i

INFORMATION NOTE

In the above instantiation, the LAD_Mux_Reset unit has been connected to the fourth element of the LAD_Bus signal. As stated above, this element must not be shared with another client.

8.4.2.3 LAD_Mux_RegFile

The component declaration for the LAD_Mux_Regfile is shown in the following code listing:

```
component LAD_Mux_RegFile is
  generic
  (
    Base  : std_logic_vector(15 downto 0) := x"0000";
    L2Num : natural := 0
  );
  port
  (
    Kclk  : in    std_logic;
    LAD   : inout LAD_Mux;
    Regs  : inout LAD_Mux_register_vector(0 to 2**L2Num-1)
  );
end component;
```

The LAD_Mux_RegFile provides an LAD accessible register file on a PE. Each register in the file is 32 bits.

The Regs port presents the values in the register file to the PE. The *Regs.Data_In* field is an output from the LAD_Mux_RegFile that provides the value written to the register file from the host over the LAD bus. The value presented to the LAD bus when the host attempts to read the register is retrieved from the *Regs.Data_Out* field. *Regs.Data_Out* is an input to the LAD_Mux_RegFile unit. To configure a register that can read the values written by the host from the PE, connect the *Regs.Data_In* field to the *Regs.Data_Out* field.

The Regs.Strobe field is an output from the LAD_Mux_RegFile. It presents a single *K_Clk* pulse when the value of the corresponding register is updated by the host.

Other than Base, the LAD_Mux_RegFile has one additional generic:

- **L2Num**: The number of registers is controlled by the *L2Num* variable. The number of registers in the file is *2**L2Num*. It occupies LAD addresses from *Base* to *Base+2**L2Num-1*.

8.4.2.4 LAD_Mux_CRegFile

The component declaration for the LAD_Mux_CRegFile is shown in the following code listing:

```
component LAD_Mux_CRegFile is
  generic
  (
    Base  : std_logic_vector(15 downto 0) := x"0000";
    L2Num : natural := 0
  );
  port
  (
    Kclk  : in    std_logic;
    LAD   : inout LAD_Mux;
```

```
        Rclk  : in    std_logic;
        Regs  : inout LAD_Mux_register_vector(0 to 2**L2Num-1)
      );
    end component;
```

The LAD_Mux_CRegFile client provides the same services as the LAD_Mux_RegFile, except that it reliably crosses clock domains. The outputs from the Regs record are updated synchronously with respect to the clock connected to the *Rclk* port.

8.4.2.5 LAD_Mux_BlockRAM

The component declaration for the LAD_Mux_BlockRAM is shown in the following code listing:

```
component LAD_Mux_BlockRAM is
  generic
  (
    Base : std_logic_vector(15 downto 0) := x"0000"
  );
  port
  (
    Kclk  : in    std_logic;
    LAD   : inout LAD_Mux;

    Mclk  : in    std_logic;
    Addr  : in    std_logic_vector(7 downto 0);
    Write : in    std_logic;
    WData : in    std_logic_vector(31 downto 0);
    RData :   out std_logic_vector(31 downto 0)
  );
end component;
```

The LAD_Mux_BlockRAM, whose component declaration is shown above, instantiates two dual-ported block RAMs with one port connected to the LAD bus and one port presented for use on the PE. The RAM is 32 bits wide and 256 words deep. *Mclk*, *Addr*, *Write*, and *WData* are inputs to the LAD_Mux_BlockRAM. *RData* is an output. Since this unit is based on Virtex BlockRAM, Reads are performed synchronously. In other words, when an address is presented on the Addr field, the value of that address from the memory is not valid until after the next clock edge. The unit occupies address space from Base to Base+255.

8.4.2.6 LAD_Mux_BlockRAM64

The component declaration for the LAD_Mux_BlockRAM64 is shown in the following code listing:

```
Component LAD_Mux_BlockRAM64 is
  generic
  (
    BASE : std_logic_vector(15 downto 0) := x"0000"
  );
  port
  (
    Kclk       : in    std_logic;
    LAD        : inout LAD_Mux;

    Mclk       : in    std_logic;
    Addr       : in    std_logic_vector(7 downto 0);
    Write_Low  : in    std_logic;
    Write_High : in    std_logic;
    WData      : in    std_logic_vector(63 downto 0);
    RData      :   out std_logic_vector(63 downto 0)
  );
end component;
```

The LAD_Mux_BlockRAM64, whose component declaration is shown above, instantiates four dual-ported BlockRAMs with one port connected to the LAD bus and one port presented for use on the PE. The RAM is 64 bits wide and 256 words deep. *Mclk*, *Addr*, *Write_Low, Write_High*, and *WData* are inputs to the LAD_Mux_BlockRAM. *RData* is an output. Since this unit is based on Virtex BlockRAM, Reads are performed synchronously. In other words, when an address is presented on the Addr field, the value of that address from the memory is not valid until after the next clock edge. The unit occupies address space from Base to Base+512.

8.4.2.7 LAD_Mux_Arb

The component declaration for the LAD_Mux_Arb is shown in the following code listing:

```
component LAD_Mux_Arb is
  generic
  (
    Register_Mux : boolean := false
  );
  port
  (
    Multi  : inout LAD_Mux_vector;
    Single : inout LAD_Mux;
    K_Clk  : in std_logic
  );
end component;
```

The LAD_Mux_Arb provide a means of taking a single LAD_Mux_Bus client and creating multiple LAD_Mux_vectors. The arbiter allows the user to create hierarchical designs and pass a single LAD_Mux_Bus down to the lower levels of the hierarchy and still use the LAD Mux Components within the lower levels. The

generic Register_Mux should be set to False to meet the timing requirements of a read on the LAD Bus.

8.4.2.8 Assigning the BASE to the LAD Mux Components

Most elements have a BASE generic that is used to map the component into each PE's LAD bus address space. Each component has a certain number of addresses it uses.

Table 8-19 shows each LAD Mux component, the number of address spaces each component consumes, and the mask created for each component.

Table 8-19: LAD Mux Address Spaces

Component	# Address Spaces	Mask
LAD_Mux_IF	0	N/A
LAD_Mux_Reset	1	0x7FFF
LAD_Mux_RegFile	2^{L2Num}	See Table 8-20
LAD_Mux_CRegFile	2^{L2Num}	See Table 8-20
LAD_Mux_BlockRAM	256	0x7F00
LAD_Mux_BlockRAM64	512	0x7E00
LAD_Mux_Arb	0	N/A

The mask for the LAD_Mux_RegFile and the LAD_Mux_CRegFile is automatically calculated from the L2Num generic. The mask starts out as 0x7FFF and for each L2num the least significant bits become zero. See the table below for examples:

Table 8-20: LAD Mux Regfile Masks

L2Num	# Registers	Generated Mask
0	1	0x7FFF
1	2	0x7FFE
2	4	0x7FFC
4	16	0x7FF0

When selecting a base address for the component, the "Mask" must be taken into consideration. If the address on the LAD bus "AND'ed" with the mask is equal to the base, the component considers that LAD bus transaction to be within the component's address space. The examples below will help when deciding on a base address.

8.4.2.8.1 LAD_Mux_Reset

Since this component only uses one address it can be placed at any unused location in the PE's address space.

8.4.2.8.2 LAD_Mux_RegFile and LAD_Mux_CRegFile

These components use 2^{L2Num} number of address. If the L2Num is 0, the component uses only one address and it can be placed at any unused location in the PE's address space. If L2Num is 1, two registers are now used.

Table 8-21: Example Base Address Table for RegFiles

Case	L2Num	Base Address	Mask	Produces Registers at	Correct/Incorrect
1	1	0x1001	0x7FFE	0x1001	Incorrect
	1	0x1000	0x7FFE	0x1000, 0x1001	**Correct**
2	2	0x2152	0x7FFC	None	Incorrect
	2	0x2150	0x7FFC	0x2150 – 0x2153	**Correct**

Case #1

Incorrect Base Choice:
The first LAD bus address will be 0x1001. When the LAD Address is AND'ed with the mask, the result will be 0x1001. This matches the base and the component considers this inside its address range. However, the second LAD bus address will be 0x1002. When the LAD Address is AND'ed with the mask, the result will be 0x1000. This doesn't match the base and the transaction is not considered to be within range, so the write to the second register is not performed.

Correct Base Choice
The first LAD bus address will be 0x1000. When the LAD Address is AND'ed with the mask, the result will be 0x1001. This matches the base and the component considers this inside its address range. The second LAD bus address will be 0x1001. When the LAD Address is AND'ed with the mask the result will be 0x1000. This also matches the base and the component considers this inside its address range.

Case #2

Incorrect Base Choice:
The first LAD bus address will be 0x2152. When the LAD Address is AND'ed with the mask, the result will be 0x2150. This doesn't match the base and the transaction is not considered to be within range, so the write to the second register is not performed. This will be true for all cases with this base.

Correct Base Choice

The first LAD bus address will be 0x2150. When the LAD Address is AND'ed with the mask, the result will be 0x2150. This matches the base and the component considers this inside its address range. The second LAD bus address will be 0x2151. When the LAD Address is AND'ed with the mask the result will be 0x2150. This also matches the base and the component considers this inside its address range. The next two addresses will produce a result of 0x2150 when the LAD address is AND'ed with the mask. Therefore all four registers will be accessible.

8.4.2.8.3 LAD_Mux_BlockRAM and LAD_Mux_BlockRAM64

The LAD_Mux_BlockRAM uses 256 addresses, while the LAD_Mux_BlockRAM64 uses 512 addresses. The LAD_Mux_BlockRAM64 is similar to the LAD_Mux_BlockRAM except it uses a mask of 0x7E00 instead of 0x7F00. The addressing of the LAD_MuxBlockRAM is described below:

Table 8-22: Example Base Address Table for LAD_Mux_BlockRAM

Base Address	Mask	Produces Registers at	Correct/Incorrect
0x1130	0x7F00	None	Incorrect
0x1100	0x7F00	0x1100 - 0x11FF	**Correct**

<u>Incorrect Base Choice:</u>
The first LAD bus address will be 0x1130. When the LAD Address is AND'ed with the mask, the result will be 0x1100. This doesn't match the base and the transaction is not considered to be within range. All subsequent LAD addresses from 0x1131 to 0x122F will have the same result, i.e., no address will match the base. (When address 0x1200 – 0x122F is AND'ed with the mask—the result will be 0x1200 instead of 0x1100.)

<u>Correct Base Choice</u>
In this case, any address from 0x1100 – 0x11FF, when AND'ed with the mask, will result in 0x1100, therefore making all addresses accessible.

The LAD_Mux_BlockRAM64 is similar to LAD_Mux_BlockRAM; however, its mask is 0x7E00 instead of 0x7F00. (Please use Table 8-22 and the examples above to calculate correct base selections for the LAD_Mux_BlockRAM64.)

8.4.3 Mem_Mux Library

The Mem_Mux Library defines a standard way of multiplexing memory ports between different user logic modules contained in the PE. The Mem_Mux Library provides a memory server that is used in place of the standard memory interface in the PE. The server is capable of providing memory access to multiple clients. The clients must obey a simple protocol when interacting with the memory server.

Compliant clients can be designed without knowledge of which other clients may be attached to the memory port, which promotes reuse of components between designs and allows components to be easily migrated between different memory ports.

The Mem36_Mux Library contains a standard client that allows the host to access 36-bit memory through the LAD bus. A sample instantiation of a Mem_Mux server is shown below:

```
U_Mem_Mux0_Priority : Mem36_Mux_Priority_IF
  port map
 (
  Mclk => Clocks_In.M_Clk,
  Reset => Global_Reset,
  Pads => Pads. MemPort0,
  Clients => Mem_Mux0
 );
```

This is the instantiation of the U_Mem_Mux0_Priority memory port from the PE template. *Mclk*, *Reset*, and *Pads* are all connected to the appropriate resources. The Clients port is a vector of records that connects the multiplexed clients to the server, which is the same approach used with the LAD_Mux server.

Clients are connected to the Mem_Mux server by assigning them an unused element of the Mem_Mux vector. Clients cannot share elements of the Clients vector.

i

INFORMATION NOTE
All available elements of a Mem_Mux_vector must be connected to a client. Unconnected elements will cause mapping failures in Synplify®.

8.4.3.1 Memory Arbitration Schemes

Two types of memory arbitration schemes can be used, both of which are described below.

8.4.3.1.1 Priority Arbitration

Mem_Mux servers that use priority arbitration can resolve multiple, simultaneous requests for access to memory. During a cycle, the client requesting access who has been assigned the lowest numbered element of the Clients vector is granted access over the others. It is possible, therefore, for a single high priority client to monopolize memory access. Memory access patterns must be carefully designed so that lower priority clients have enough memory bandwidth available to perform their functions.

The Priority Arbiter has generics that can be set by the user upon instantiation of the component. Please refer to Table 8-23: Priority Arbiter Generics, for a complete description of these generics. Please note that the Num_Akk_FIFOs generic is not available in the PE0 VHDL model.

Table 8-23: Priority Arbiter Generics

Generic Name	Values	Description
Avoid_Overflow	True/ False	This adds logic that guarantees that an arbiter's internal AkkFIFO does not overflow. If the user knows that the latency from the MemMux to memory and back is less than the FIFO depth, this generic can be set to false to avoid the additional logic and timing troubles.
Num_Akk_FIFOs	Integer	This sets the number of FIFOs used to store previous Akks for routing return values to their destinations. The value 0 is a special case for backwards compatibility, in which a single SRLFifo is used (worse timing) instead of an SRLFDFifo. Values greater than 0 include that many SRLFDFIFOs. Each FIFO can store 17 values, so the maximum number of outstanding requests which can be stored is 17*Num_Akk_FIFOs, or 16 when Num_Akk_FIFOs=0.
Rotate_Priority	True/ False	When arbitrating between multiple clients, the priority can either be fixed, in which case client 0 will always have the highest priority, or it can change based upon who was acknowledged previously.
Sticky_Priority	True/ False	With a rotating priority, the client that was acknowledged on the previous clock cycle may either be the highest priority on this clock cycle (TRUE), or it may become the lowest priority (FALSE). When the previously acknowledged client becomes the highest priority, that client can monopolize the bus by performing requests on every clock cycle, thus making the priority stick at that client. If Rotating_Priority is FALSE, the value of Sticky_Priority is irrelevant.
Registered_Reqs	True/ False	An arbiter may either react in combination with respect to the requests being made, or it may register these requests and process them a clock cycle later. The latter mode is better for timing reasons if not all client requests already come from registers.
Registered_Akks	True/ False	Once a decision is made on which client to “akk,” the akk may either be presented immediately, or it may be delayed one clock cycle. Delaying akks is usually better for timing reasons since the akks typically go straight in to a multiplexer. The disadvantage of registering Reqs and Akks is the response time with which a client is acknowledged. Each time a client asserts or deasserts its request line, there may be up to a two clock cycle period during which the memory remains idle because an inactive client is still being acknowledged. If only one of Reqs and Akks is registered, this dead time is reduced to one clock cycle, maximum. Usually, this effect is never an issue, but if it is, using rotating non-sticky priorities and placing Mem*_Mux_IFIFOs in line will alleviate these problems.
Register_Data	True/ False	This generic is used to add registers between the IOB registers and the interfaces that connect to the priority arbiters. When using this generic, it is important to remember that the delay when performing a read is increased by two clock cycles.

8.4.3.1.2 Fair Arbitration

Another Mem_Mux server with a different arbitration scheme is also available— the fair arbiter. The fair arbiter simply rotates access to memory among all of its clients. The more clients present in the system, the less memory bandwidth each is granted. Each client is granted an equal slice of memory bandwidth whether or not it will actually be used.

The Fair Arbiter has several generics that can be set by the user upon instantiation of the component. Please refer to Table 8-24 for a complete description of these generics.

Table 8-24: Fair Arbiter Generics

Generic Name	Values	Description
Avoid_Overflow	True/ False	This adds logic that guarantees that an arbiter's internal AkkFIFO does not overflow. If the user knows that the latency from the MemMux to memory and back is less than the FIFO depth, this generic can be set to false to avoid the additional logic and timing troubles.
Register_Data	True/ False	This generic is used to add registers between the IOB registers and the interfaces which connect to the priority arbiters. When using this generic it is important to remember that the delay when performing a read is increased by two clock cycles.

The WILDSTAR™ board uses two similar Mem_Mux components. The Mem64_Mux is used when connecting to 64-bit wide memories. The Mem32_Mux is used when connecting to the 32-bit wide memory. The two components are described below.

8.4.3.2 Mem36_Mux Control

The Mem36_Mux record definitions are shown in Table 8-25:

Table 8-25: Mem32_Mux Record Signals

Signal Name	Width	Dir	Description
Addr	36	Input	User interface to the Memory bus address
Write	1	Input	Write select signal; indicates a write cycle when ‘1’ or read cycle when ‘0’
Data_In	36	Output	User interface to the data from the 36-Bit memory port.
Data_Out	36	Input	User interface to the data to the 36 bit memory port.
Data_Valid	1	Output	Data Valid signal which is high when valid data is present from a 36-bit memory port.
Req	1	Input	Indicate that an operation, either read or write, to the memory is valid.
Akk	1	Output	Indicates the memory interface as accepted a requested transaction and is attempting to complete it.

When a client wishes to perform a Write cycle, the address of the operation on the *Addr* port should be presented. The outgoing data is placed on the *Data_Out* lines. Asserting the *Write* line indicates a Write. Finally, asserting the *Req* line indicates that the operation being presented by the client is valid. The client must continue to

present its memory operation until it samples *Akk* high on a rising clock edge. A high *Akk* signal indicates that the memory interface has accepted the transaction and will attempt to complete it.
Situations where the memory operation is never completed indicate possible design errors or application malfunctions. The Mem36_Mux server provides arbitration between its various clients. Some arbitration schemes can be misused to produce situations where operations may not be able to complete. If, for instance, a priority arbitration scheme is being used and a high priority client never relinquishes control of the memory, lower priority clients can never complete their memory transactions.

Mem36_Mux clients should make no assumptions about the behavior of the *Akk* signal. *Akk* may be asserted or deasserted at any time. *Akk* may be asserted whether or not a request is being made. In some cases, the priority arbiter that comes with the library, *Akk,* is a combination function that includes the client's *Req*. In other cases, like the fair arbiter that comes with the library, the assertion of *Akk* is completely decoupled from the client. The fair arbiter simply provides the *Akk* to clients in round-robin fashion. Even if a designer knows beforehand which arbiter will be used to comply with the Mem36_Mux specification, and thereby be reusable within the context of the Mux Library, clients must be capable of interoperating with an arbitrary arbiter.

The timing for a Read cycle is illustrated in the following timing diagram. To read from memory, a client should assert the desired address and deassert the *Write* line. Asserting the *Req* line indicates that the current request is valid. As with a Write request, the Read request is accepted when *Akk* is sampled high on a rising clock edge. No assumptions should be made about the behavior of the *Akk* line.

After a Read request has been accepted, the corresponding data from memory will be returned on the *Data_In* lines accompanied by an asserted *Data_Valid* line. Returning data is guaranteed to arrive in the order in which it was requested. Mem36_Mux clients may make no assumptions about the latency between the acceptance of a request and the arrival of the requested data.

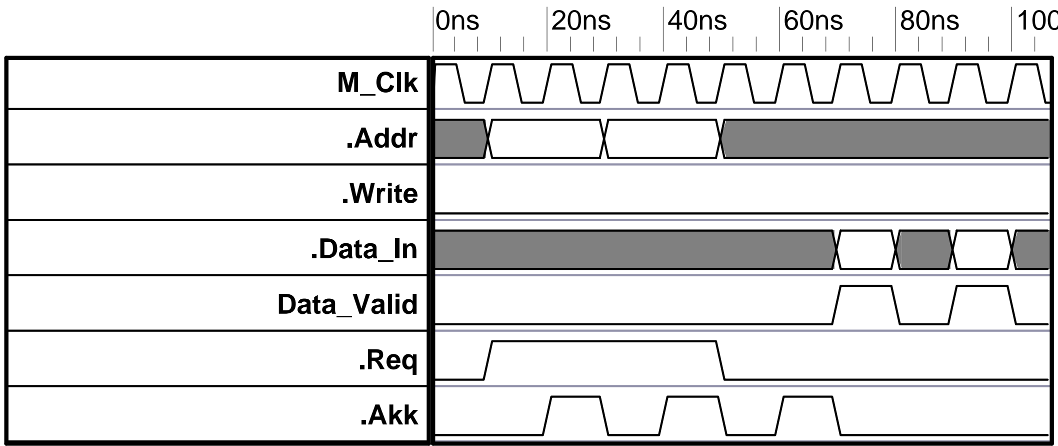


Figure 8-7: Mem36_MuxRead Cycle

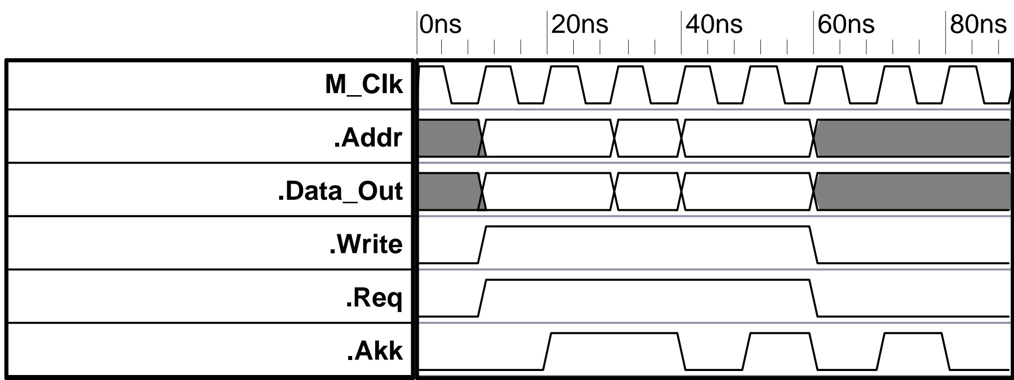


Figure 8-8: Mem36_MuxWrite Cycle

8.4.4 Programmed I/O Memory Bridge

8.4.4.1 LAD_Mem36_Bridge

The LAD_Mem36_Bridge provides LAD access to the Mem36_Mux memory ports.

The LAD_Mem36_Bridge uses a Base generic used to map the component into the LAD bus address space. (The Base generic is described here; additional generics are described in the sections below.) If the address on the LAD bus AND'ed with the Mask is equal to the Base, the component considers that LAD bus transaction to be within the component's address space. The LAD_Mem36_Bridge consumes 512 LAD address spaces. The component declaration for the LAD_Mem36_Bridge is shown in the following code listing:

```
component LAD_Mem36_Bridge is
 generic
 (
```

```
    BASE : std_logic_vector(15 downto 0) := x"0000
   );
  port
  (
    Kclk : in std_logic;
    Mclk : in std_logic;
    LAD : inout LAD_Mux;
    Mem : inout Mem36_Mux
   );
end component;
```

Using an indirect addressing scheme, the LAD_Mem36_Bridge provides LAD access to a Mem36_Mux memory port. Data headed for memory is buffered up from the LAD bus in an on-PE Block RAM then copied into memory. Data being retrieved from memory is copied from memory to the Block RAM and then read out across the LAD Bus. The Mux Library provides C-based memory access functions that serve as a simplified mechanism for interacting with memory via the bridge. The LAD_Mem36_Bridge occupies addresses from Base through Base+255.

The LAD_Mem36_Bridge allows the user to write and read a "tag" value to the upper four bits of the 36 bit memory. During a write to the memories, the "tag" will be appended to the data supplied by the host. The "tag" value is stored in the lowest nibble of fourth LAD Mux Register Vector (BASE + 0x103). During a read, the Mem_Select signal selects which 32-bit value to read from the memories.

When Mem_Select is low the lower 32 bits of data is sent back to the host; this read will not contain "tag" information. When Mem_Select is high, the upper 32 bits of the data is read from the host; this read will contain the "tag" and the upper 28 bits of data. The Mem_Select signal is stored in the least significant bit of the third LAD Mux Register Vector (BASE + 0x102).

8.4.4.2 Assigning the BASE to the LAD Memory Bridges

The LAD memory bridges have a BASE generic used to map the component into each PE's LAD bus address space.

When selecting a base address for the bridges, the "mask" must be taken into consideration. If the address on the LAD bus AND'ed with the mask is equal to the base, the component considers that LAD bus transaction to be within the component's address space. The examples below will help when deciding on a base address.

8.4.4.2.1 LAD_Mem32_Bridge

As noted, the LAD_Mem36_Bridge uses 512 addresses. The table below shows the address map of the LAD_Mem36_Bridge. The base for this example is 0x0; therefore the addresses consumed by this component are 0x0 - 0x200.

Table 8-26: Address in the LAD_Mem36_Bridge

Addresses	Consumed By
0x000-0x0FF	LAD_Mux_BlockRAM
0x100-0x101 0x102-0x1FF	LAD_Mux_CRegfile w/L2Num = 1 Reserved

The example below will help when deciding on a base address:

Table 8-27: Example Base Address Table for the Mem36 Bridge

Base Address	Mask	Correct/Incorrect
0x1150	0x7F00	Incorrect
0x1000 0x1100	0x7F00 0x7F00	**Correct** **Correct**

Incorrect Base Choice:
Because this component contains a LAD_Mux_BlockRAM, the same problems described in Section 8.4.2.8.3 will occur. Therefore, no information from the LAD bus will be written or read to/from the memories.

Correct Base Choice
When the address on the LAD bus ranges from 0x1000 – 0x10FF and is AND'ed with the mask, the result will be 0x1000. Data from the host will be allowed to write/read to/from the Block RAM. The Mem Bridge also contains a LAD_Mux_CRegFile, which sets at address "BASE+0x100", in this case 0x1300. Since the mask is 0x7F00, when the LAD address is AND'ed with the mask, the result will be 0x1100. This matches "BASE+0x100" and the component considers this inside its address range.

0x1100 is also an acceptable "BASE" address selection for the same reasons as above. When using two or more LAD_Mem36_Bridges, the base addresses must be at least 0x200 apart, or the addresses from one component will overlap and will cause erroneous behavior in the design.

8.4.5 DMA_Mux Library

The DMA_Mux Library provides a reusable set of components allowing the user to create fully functional LAD bus interfaces with minimum VHDL coding. Figure 8-77 illustrates the General DMA Mux Library Scheme.

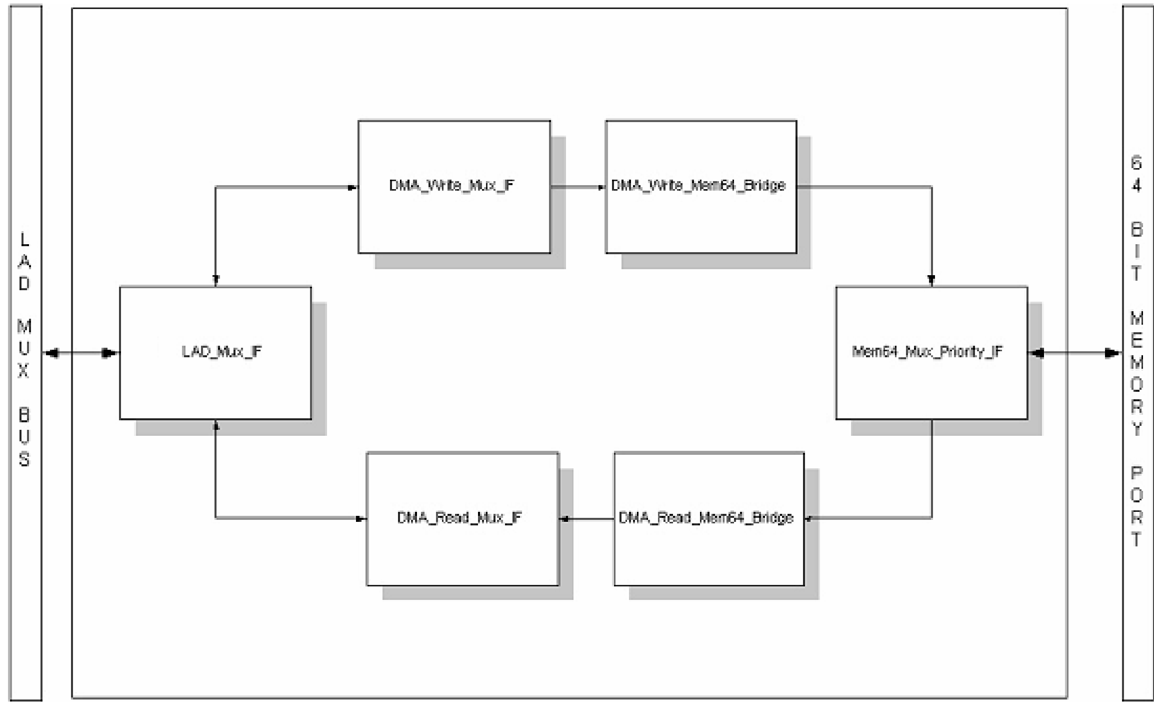


Figure 8-7: General DMA Mux Library Scheme

8.4.5.1.1 LAD_DMA_Write_Mux_IF

The component declaration for the LAD_DMA_Write_Mux_IF is shown in the following code listing:

```
component LAD_DMA_Write_Mux_IF is
  port
  (
    Kclk        : in    std_logic;
    LAD         : inout LAD_Mux;
    Rclk        : in    std_logic;
    Reset       : in    std_logic;
    DMA         : inout LAD_DMA_Write_Mux_vector
  );
end component;
```

The LAD_DMA_Write_Mux_IF component provides the user with DMA data from the host system and controls the DMA_Wstat status flags. The DMA port signals are described in the following table:

Table 8-28: LAD_DMA_Write_Mux_Vector Record Signals

Signal Name	Width	Dir	Description
Data_Valid	1	Output	User interface which will be '1' when valid data is "Pulled" from the FIFO.
Pull	1	Input	Active high signal used to pull data from the FIFO.
Data_In	32	Output	User interface to the input data from the LAD bus. The data will be 32 bits.
Fifo_Status	5	Output	User interface indicating the status of both FIFOs.
Fifo_Empty	1	Output	User interface signal, which indicates the empty status of the FIFO.
Fifo_Full	1	Output	User interface signal, which indicates that either the upper or lower FIFO has reached the ¾ full mark. The ¾ mark is used to prevent overrunning the input FIFOs

The LAD port is a unique LAD_Mux_Vector, which is described in Table 8-18: LAD_Mux Record Signals. The Rclk signal is user-defined except when the LAD_DMA_Write_Mux_IF is connected to a DMA_Write_Bridge—in this case, Rclk must be Mclk. The Reset signal should be connected to the Global_Reset output of a LAD_Mux_Reset component.

The LAD_DMA_Write_Mux_Vector provides the user with interactive control signals to and from the Block RAM FIFOs. The Data_Valid indicate when data is valid on the Data_In bus. When the Pull signal is asserted and valid data is in the FIFO, valid data will be available on the next clock cycle. To avoid an overflow, the Pull signal is "ANDed" with the Fifo_Empty signal. The Fifo_Status lines indicate when the FIFO is empty, 1 DWORD to ¼ full, ¼ to ½ full, ½ to ¾ full, and ¾ full to full.

Only one LAD_DMA_Write_Mux_IF can be instantiated in a design, although multiple LAD_DMA_Write_Mux_Vectors can be connected to this interface. The LAD_DMA_Write_Mux_Vector follows the same rules as a LAD_Mux_Vector, in that each vector must have a unique connection and no connections can be skipped.

The LAD_DMA_Write_Mux_IF can be connected directly into the PE, or it can be used in conjunction with a LAD_DMA_Write_Mem36_Bridge or a LAD_DMA_Write_Mem64_Bridge to write data to memory.

8.4.5.1.2 LAD_DMA_Read_Mux_IF

The component declaration for the LAD_DMA_Read_Mux_IF is shown in the following code listing:

```
component LAD_DMA_Read_Mux_IF is
  port
  (
    Kclk        : in    std_logic;
    LAD         : inout LAD_Mux;
    Rclk        : in    std_logic;
    Reset       : in    std_logic;


    DMA         : inout LAD_DMA_Read_Mux_vector
  );
```

```
end component;
```

The LAD_DMA_Read_Mux_IF component sends DMA data to the host system and controls the DMA_Rstat status flags. The DMA port signals are described in Table 8-29.

Table 8-29: LAD_DMA_Read_Mux_Vector Record Signals

Signal Name	Width	Dir	Description
Data_Out	64	Input	User interface to the DMA data.
Push	1	Input	User interface which will be '1' when valid data is "Pulled" from the FIFO.
Fifo_Status	5	Output	User interface which indicate the status of the FIFO.
Fifo_Empty	1	Output	User interface signal, which indicates the empty status of the FIFO.
Fifo_Full	1	Output	User interface signal, which indicates that either the FIFO has reached the ¾ full mark. The ¾ mark is used to prevent over running the input FIFO.

The ports on LAD_DMA_Read_Mux_IF consist of Kclk, LAD, Rclk, Reset and DMA. The Kclk port must be the K_Clk_2x signal from the Clock_Std_IF. The LAD port is a unique LAD_Mux_Vector, which is described in Table 8-18: LAD_Mux Record Signals. The Rclk signal is user defined except when the LAD_DMA_Write_Mux_IF is used with a DMA_Write_Bridge; in this case, Rclk must be Mclk. The Reset signal should be connected to the Global_Reset output of a LAD_Mux_Reset component.

The LAD_DMA_Read_Mux_Vector provides the user with interactive control signals to and from the Block RAM FIFOs. The Push signal is used with the Data_Out signal to "Push" data onto the data from the PE. The Fifo_Status lines are taken from the lower FIFO and indicate when the FIFO is empty, 1 DWORD to ¼ full, ¼ to ½ full, ½ to ¾ full, and ¾ full to full. The Fifo_Empty is high when there are no more DWORDs left in the FIFO. The Fifo_Full signal is high when either FIFO becomes ¾ full.

Only one LAD_DMA_Read_Mux_IF can be instantiated in a design, although multiple LAD_DMA_Read_Mux_Vectors can be connected to this interface. The LAD_DMA_Read_Mux_Vector follows the same rules as a LAD_Mux_Vector, in that each vector must have a unique connection and no connections can be skipped in the vector.

The LAD_DMA_Read_Mux_IF can be connected directly to the PE. It also can be used in conjunction with a LAD_DMA_Read_Mem36_Bridge or a LAD_DMA_Read_Mem64_Bridge to read data from memory.

The DMA_Read_Mux_IF has one generic:

- **BYTE_SWAP**: This Boolean generic is used to byte swap the data before it is presented on the LAD Bus.

8.4.6 DMA Memory Bridges

The DMA Memory Bridges are used to transfer DMA data to and from the host system to the local memories on WILDSTAR™ boards. The component instantiation for the LAD_DMA_Write_Mem36_Bridge and the LAD_DMA_Read_Mem36_Bridge are shown in the following code listing:

```
U_DMA_Write_Bridge : LAD_DMA_Write_Mem36_Bridge
  generic map
  (
    BASE        => x"1000",
    RELAOD_EN   => FALSE
  )
  port map
  (
    Kclk        => Clocks_In.K_Clk_2x,
    Reset       => Global_Reset,
    LAD         => LAD_Mux_Bus(4),

    Mclk        => Clocks_In.M_Clk,
    Mem         => Mem_Mux_0(0),
    DMA         => DMA_Write_Mux_Bus(0),
    Status      => DMA_Status_0
  );

U_DMA_Read_Bridge : LAD_DMA_Read_Mem36_Bridge
  generic map
  (
    BASE        => x"1200",
    RELAOD_EN   => FALSE
  )
  port map
  (
    Kclk        => Clocks_In.K_Clk_2x,
    Reset       => Global_Reset,
    LAD         => LAD_Mux_Bus(4),

    Mclk        => Clocks_In.M_Clk,
    Mem         => Mem_Mux_0(1),
    DMA         => DMA_Read_Mux_Bus(0),
    Status      => DMA_Status_1
  );
```

The ports on LAD_DMA_Write_Mem36_Bridge and LAD_DMA_Read_Mem36_Bridge are similar, each consisting of three generics and the ports Kclk, Reset, LAD, Mclk, Mem, DMA, and Status. The BASE generics are necessary in all DMA bridges because they contain a single LAD_Mux_CRegfile. The CRegfile is used to control the start and stop memory addresses.

Kclk must be the Clocks_In.K_Clk_2x signal from the Clock_Std_IF. The Reset signal should be connected to the Global_Reset output of a LAD_Mux_Reset component. The LAD port is a unique LAD_Mux_Vector described in Table 8-18: LAD_Mux Record Signals.

The Mclk signal is the memory interface clock. The Mem port is either a Mem36_Mux Vector or a Mem64_Mux Vector. The DMA port is described in section 8.4.6.2.

Each DMA memory bridge contains a status record that can be used in the PE to determine if a DMA operation is being performed and what percentage of the DMA operation has been completed. The DMA port signals are described in Table 8-30.

Table 8-30: LAD_DMA_Status Record Signals

Signal Name	Width	Dir	Description
Start_Addr	32	Output	Starting address of memory address counter.
Stop_Addr	32	Output	Stopping address of memory address counter.
Curr_Addr	32	Output	Current address of the DMA operation. This number can be subtracted from the Stop_Addr to determine how many QWORDS are left in the current DMA operation.
Busy	1	Output	Active high signal indicating an active DMA operation.
Init_Xfer	1	Input*	Used with the RELOAD_EN generic. See the individual Memory Bridge definition for further explanation.

* In the DMA_Read_Bridges, this signal is driven low if RELOAD_EN is set to false, otherwise it must be driven by the user.

8.4.6.1 DMA Write Bridges

The LAD_DMA_Write_Mem36_Bridge is used in conjunction with a LAD_DMA_Write_Mux_IF to take data from the host system and write it to memory.

As stated earlier, the Base generic is necessary in the DMA bridges because the bridges contain a single LAD_Mux_CRegfile with a total of four registers. The CRegfile is used to control the start and stop memory addresses. The following excerpt of ANSI C code shows the necessary steps in the host program for setting up a memory write bridge and completing a DMA transfer:

```
dReadAddr[0] = 0;
dReadAddr[1] = 0x1500;

/* Set start and stop addresses */
printf ("\tSetting Starting Address ... ");
rc = WS_WritePeReg( WS_Board, 0x0, 0x1000, 2, dReadAddr );
if ( rc != WS_SUCCESS )
{
 printf( "ERROR %d: %s\n", rc, WS_ErrorString( rc ) );
 WS_DmaMemFree ( WS_Board, WriteBuffer );
 return(rc);
}
 printf ("Done\n");

/* DMA Data to the selected PE /*
```

```
printf ("\tDMAing Data to PE[0] ... " );
rc = WS_DmaWrite ( WS_Board, 0x0, 0x1500, WriteBuffer, &Cnt, 0x0,
0x0 );
if ( rc != WS_SUCCESS )
{
 printf( "ERROR %d: %s\n", rc, WS_ErrorString( rc ) );
 WS_DmaMemFree ( WS_Board, WriteBuffer );
 return(rc);
}
printf ("Done\n");
```

In the lines above, there are only two API calls. The first API, WS_WritePeReg, is used to set the start and stop memory addresses. In this case the start address is 0 and the ending address is 0x1500. The second API, WS_DmaWrite, is used to transfer the data from the host system to the memory. In this case, WriteBuffer would contain 0x1500 DWORDS of data. If the calls are successful, the “rc” values of the API calls will be WS_SUCCESS.

The write bridges contain the generic RELOAD_EN. This generic allows the start address to be reset after a DMA transfer is completed. Although several write bridges can be instantiated at a time, only one DMA write bridge should be active simultaneously. For this reason, even if the RELOAD_EN generic is set to true, the address counter will not reset unless the Status.Init_Xfer signal is high at the end of a DMA operation. The Status.Init_Xfer line to a particular bridge can be held high during a DMA write operation, but the address will only be reset when the memory address reaches the count provided by the programmed I/O write. This action is necessary so the data coming from the LAD_DMA_Write_Mux_IF only goes to one bridge. The bridges use the fifo_empty signal to determine when data should be pulled from the write mux interface. The component declaration for the LAD_DMA_Write_Mem36_Bridge is shown in the following code listing:

```
Component LAD_DMA_Write_Mem36_Bridge is
 generic
 (
 BASE : std_logic_vector(15 downto 0) := x"0000";
 RELOAD_EN : boolean := FALSE
 );
 port
 (
 Kclk : in std_logic;
 Reset : in std_logic;
 LAD : inout LAD_Mux;
 Mclk : in std_logic;
 Mem : inout Mem36_Mux;
 DMA : inout LAD_DMA_Write_Mux;
 Status : inout LAD_DMA_Status
 );
end component;
```

8.4.6.1.1 LAD_DMA_Write_Mux_Arb

The component declaration for the LAD_DMA_Write_Mux_Arb is shown in

the following code listing:

```
component LAD_DMA_Write_Mux_Arb is
 generic
 (
 Register_Mux : boolean := false
 );
 port
 (
 Multi : inout LAD_DMA_Write_Mux_vector;
 Single : inout LAD_DMA_Write_Mux;
 Rclk : in std_logic
 );
end component;
```

The LAD_DMA_Write_Mux provides a means of taking a single LAD_DMA_Write_Mux client and creating multiple LAD_DMA_Write_Mux_Vectors. The arbiter allows the user to create hierarchical designs and pass a single LAD_DMA_Write_Mux client down to the lower levels of the hierarchy and still use the LAD Mux Components within the lower levels. The generic Register_Mux can be set to true, if needed, to improve design timing.

8.4.6.2 DMA Read Bridges

The LAD_DMA_Read_Mem36_Bridge is used in conjunction with a LAD_DMA_Read_Mux_IF to DMA data from the memories on an I/O Card and send it to the host system. As stated earlier, the Base generic is necessary in the DMA bridges because the bridges contain a single LAD_Mux_CRegfile, with four registers.

The CRegfile is used to control the start and stop memory addresses. The following excerpt of ANSI C code shows the steps necessary to set up a memory read bridge and complete a DMA transfer:

```
dReadAddr[0] = 0;
dReadAddr[1] = 0x1500;

/* Set start and stop addresses */
printf ("\tSetting Starting Address ... ");
rc = WS_WritePeReg( WS_Board, 0x0, 0x1000, 2, dReadAddr );
if ( rc != WS_SUCCESS )
{
 printf( "ERROR %d: %s\n", rc, WS_ErrorString( rc ) );
 WS_DmaMemFree ( WS_Board, ReadBuffer );
 return(rc);
}
printf ("Done\n");
/* DMA Data from PE0 /*
printf ("\tDMAing Data from PE[0] ... " );
rc = WS_DmaRead ( WS_Board, 0x0, 0x1500, ReadBuffer, &Cnt,
0x0,
0x0 );
if ( rc != WS_SUCCESS )
{
 printf( "ERROR %d: %s\n", rc, WS_ErrorString( rc ) );
 WS_DmaMemFree ( WS_Board, ReadBuffer );
```

```
        return(rc);
      }
            printf ("Done\n");
```

In the lines above, there are only two API calls. The first API, WS_WritePeReg, is used to set the start and stop memory addresses. In this case, the start address is 0 and the ending address is 0x1500. The second API, WS_DmaRead, is used to transfer the WILDSTAR™ local memory to the host system. If the calls are successful, the “rc” values of the API calls will be WS_SUCCESS.

The Read bridges contain the generic RELOAD_EN. This generic allows the start address to be reset after a DMA transfer is completed. Although several read bridges can be instantiated at a time, only one DMA read bridge can be active at a time. The read bridge acts as a prefetch unit for the LAD_DMA_Read_Mux_If. As soon as the stop address doesn’t match the start address, data is retrieved from memory and stored in the FIFOs in the read interface. For this reason, the PE assumes responsibility for the reset of the start counter.

Once the PE drives Status.Init_Xfer, the start and stop counters will be reset to their original values written by the programmed I/O write. This action begins the prefetching of data from memory for the LAD_DMA_Read_Mux_IF. The bridges use the fifo_full signal to determine when a pause in fetching data should occur.

The component declaration for the LAD_DMA_Read_Mem36_Bridge is shown in the following code listing:

```
Component LAD_DMA_Read_Mem36_Bridge is
  generic
  (
  BASE : std_logic_vector(15 downto 0) := x"0000";
  RELOAD_EN : boolean := FALSE
  );
  port
  (
  Kclk : in std_logic;
  Reset : in std_logic;
  LAD : inout LAD_Mux;
  Mclk : in std_logic;
  Mem : inout Mem36_Mux;
  DMA : inout LAD_DMA_Read_Mux;
  Status : inout LAD_DMA_Status
  );
 end component;
```

8.4.6.2.1 LAD_DMA_Read_Mux_Arb

The component declaration for the LAD_DMA_Read_Mux_Arb is shown in the following code listing:

```
component LAD_DMA_Write_Mux_Arb is
generic
(Register_Mux : boolean := false
);
port
(
Multi : inout LAD_DMA_Read_Mux_vector;
Single : inout LAD_DMA_Read_Mux;
Rclk : in std_logic
);
end component;
```

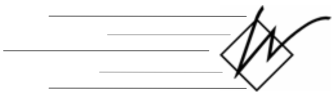
The LAD_DMA_Read_Mux provides a means of taking a single LAD_DMA_Read_Mux client and creating multiple LAD_DMA_Read_Mux_Vectors. The arbiter allows the user to create hierarchical designs and pass a single LAD_DMA_Write_Mux client down to the lower levels of the hierarchy and still use the LAD Mux components within the lower levels. The generic Register_Mux can be set to True, if needed, to improve design timing.

8.4.6.3 Assigning the BASE to the LAD Memory Bridges

The DMA Bridges each contain a LAD64_Mux_CRegFile with the L2Num set to two, meaning that there are a total of four address spaces consumed by these components.

8.4.7 Host Model

The simulated host model is used to access the PE resources of the WILDSTAR™ board via the PCI Controller. It can also be used to initialize and analyze the contents of any memory device in the WILDSTAR™ system. The simulated host model is similar to the actual host system in that the user writes a “program” that consists of standard API functions that interact with the WILDSTAR™ board. In fact, the simulated VHDL host API procedures have nearly identical structure and usage as the actual software API functions.



9. FPDP-E I/O CARD VHDL GUIDE

The VHDL design cycle consists of several simple steps that help an application developer create, simulate, synthesize, and analyze a FPDP-E system design. This chapter describes how to create a design using the FPDP-E system VHDL models.

9.1 VHDL Design Cycle

The FPDP-E VHDL model is not a standalone VHDL mode. In order to use the FPDP-E VHDL model, a motherboard such as the WILDSTAR™ /VME, FIREBIRD™ /PCI, WILDSTAR™-II /VME, or WILDSTAR™-II /PCI must be compiled in conjunction with the model.

9.1.1 FPDP-E Template VHDL Design Files

The VHDL template files, described in sections 9.1.1.2 and 9.1.1.3, are all found in the sub-directory “$IO_CARD_BASE/fpdpe_io_card/template/sim”. These files should be copied to a design project directory and used as a starting point for the design.

9.1.1.1 ModelTech™ Macro Scripts

Three macro scripts are necessary to compile the FPDP-E VHDL models. Two of the files, the **project_vcom.do** and **project_vsim.do**, are contained in the WILDSTAR™ /VME and FIREBIRD™ /PCI sub directories (Please refer to the WILDSTAR™ /VME or FIREBIRD™ /PCI hardware manual for more information.).

The third script, the **project_fpdpe_vcom.do** macro file is used to compile additional user related VHDL files. Follow the itemized instructions located in the project_vcom.do file to customize the FPDP-E simulation model.

9.1.1.2 PE VHDL Template Files

9.1.1.2.1 FPDP-E I/O PE Architecture Template

On of the two template files, **fpdpe_io_card_template_arch.vhd** or **fpdpe_io_card_pe_wsii_template_arch.vhd** should be used when creating a FPDP-E design, since these template files have been designed to contain most of the boilerplate interface logic that is needed by every design. Starting from scratch would not only take much longer, but would also increase the risk of implementing incorrect or incomplete interface logic.

9.1.1.2.2 PDP-E I/O Connector Interface Template

The **fpdpe_if_template_arch.vhd** file is used to simulate external connections to the FPDP-E I/O card.

9.1.1.2.3 FPDP-E Backplane Interface Template

The **fpdpe_backplane_if_template_arch.**vhd file is used to simulate backplane connections to the FPDP-E I/O Card.

9.1.1.3 FPDP-E Configuration Templates

The FPDP-E I/O card configuration files (**fpdpe_io_card_0_cfg.vhd** and **fpdpe_io_card_1_cfg.vhd**) must be modified to meet the needs of the design. The FPDP-E I/O elements that can be modified by the user include:

U_PE: The PE model architecture can be changed to the name used by a particular application.

U_MemPort0: (Memory bank 0): The architecture of the memory bank can be changed to “Empty” (when no memory is needed) or “Static” (when memory is needed). Also, the “MEM_SIZE” generic can be changed to match the needs of the application.

U_MemPort1: (Memory bank 1): The architecture of the memory bank can be changed to “Empty” (when no memory is needed) or “Static” (when memory is needed). Also, the “MEM_SIZE” generic can be changed to match the needs of the application.

U_MemPort2: (Memory bank 2): The architecture of the memory bank can be changed to “Empty” (when no memory is needed) or “Static” (when memory is needed). Also, the “MEM_SIZE” generic can be changed to match the needs of the application.

U_MemPort3: (Memory bank 3): The architecture of the memory bank can be changed to “Empty” (when no memory is needed) or “Static” (when memory is needed). Also, the “MEM_SIZE” generic can be changed to match the needs of the application.

U_FPDP_IF: The FPDP-E interface architecture can be changed to the name used by a particular application. The FPDP-E interface can be used to model signals coming and going through the 38-pin FPDP-E MICTOR® connector.

U_BackPlane_IF: The Backplane interface architecture can be changed to the name used by a particular application. The Backplane interface can be used to model signals coming and going through the 95/96 pin P0/P2 connector.

9.1.2 Simulating a VHDL Design

Once the VHDL design has been compiled, the design can be simulated using the ModelSim™ tool. Follow these simple steps to simulate the design:

1. Start the ModelSim™ tool.
2. In the **Macro** menu (or in the **File** menu in ModelSim™ v4.7 or older), select the **Execute Macro…** option.
3. Use the file browser to locate the design project directory.
4. Select the **project_vsim.do** compilation macro file.
5. Click on the **Open** button.

At this point, the loading messages will begin to scroll in the **ModelSim** window (or **Transcript** window in ModelSim™ v4.7 or older). If any errors occur during the loading of the design, the error message(s) will also appear in the **ModelSim**/**Transcript** window. Once the errors are corrected, repeat Steps 1 through 5 above until the design has been completely loaded and is ready for simulation. Refer to the ModelSim™ manual for details on how to run the simulation.

9.1.3 Synthesizing a VHDL Design

This section discusses how to synthesize a design using the Synplicity Synplify™ VHDL synthesis tool.

9.1.3.1 Using Template Synplify™ Project Files

The following Synplify™ project files are found in the subdirectory called "$IO_CAR_BASE/fpdpe_io_card/template/syn" and should be copied to a design project directory and used as a starting point for the design:
pe/pe.prj FPDP-E I/O Card PE project VHDL synthesis project file
for Synplify™ .

Once the PE project file has been copied, it should be modified to meet the needs of the design project. Simply follow the step-by-step instructions located inside the project file to customize it for the current design project.

9.1.3.2 Setting up Synthesis Constraints

Certain design constraints can be configured using the Synplify™ synthesis tool. Some of these constraints are located at the end of the Synplify™ project file. Other design constraints may be added directly to the VHDL code as VHDL attributes while others can be added to the Synplify™ design constraints file (*.sdc). Refer to the Synplify™ manual for more information regarding synthesis design constraints.

9.1.3.3 Running the Synplicity Synthesis Tool

Once the Synplify™ project file and design constraints have been configured, the PE design can be synthesized. Follow these simple steps to synthesize the PE or BPE design:

1. Start the Synplify™ tool.
2. Close any project window that might be open.
3. In the **File** menu, select the **Open Project…** option.
4. Use the file browser to locate the design project directory.
5. Click on the **Open** button.
6. Click on the **RUN** button.

At this point, the compiling and mapping messages will begin to appear in the project window. If any warnings or errors occur during the loading of the design, the error indicator will also appear in the project window. Click on the View Log button to review the warning and/or error messages.

Due to the comprehensive nature of the FPDP-E PE model, you will undoubtedly encounter warnings during the synthesis process. These warnings are usually related to PE I/O signals that are simply unused by the current design project. As a rule, you can ignore any warnings that do not appear in your own design files.

Once the errors (and user warnings) are corrected, repeat Steps 1 through 7 above until the design has been completely loaded and is ready for place-and-route. Refer to the Synplify™ manual for more details on how to use the Synplify tool.

9.1.4 Place-and-Routing a Design

Once the design has been synthesized and an EDIF design file has been produced, the Xilinx® Alliance Series tools can be used to place-and-route the design. The Xilinx Foundation Series tools include all of the necessary Xilinx Alliance Series tools.

9.1.4.1 Setting Environment Variables

There are two environment variables which should be added/modified to the users profile. The first is only necessary under Windows NT® systems. Click Start -> Settings -> Control Panel -> system. Click on the environment tab. Add a new variable called MAKE_MODE and set the value to UNIX.

The second variable, which should already exist, is the path variable. Add to the path the variable "$ANNAPOLIS_BASE/shared/bin". These changes/modifications are necessary for proper makefile execution.

9.1.4.2 Using Template Makefiles

The following Xilinx® Alliance Series makefile file is found in the sub-directory called "$IO_CARD_BASE/template/syn" and should be copied to a design project directory and used as a starting point for the design:

pe/makefile PE0 project makefile for Xilinx Alliance tools

Once the makefile has been copied, it should be edited so that the ANNAPOLIS_BASE macro properly reflects the FPDP-E installation directories. Be sure to use forward slashes ('/') in the path name, even if using an operating system that usually requires backslashes in a path name.
There are several other variables which must be modified in order for the makefile to produce the correct binary file for the FPDPE .

1. **DEF_PART**: must be set to match the PE part type.
2. **DEF_SPEED**: should be set to match the speed grade of the PE. IF the speed grade does not match, erroneous timing reports will be generated.
3. **DEF_ENDIAN**: This produces either a little-endian or a Big-endian binary file.

9.1.4.3 Setting up Timing Constraints

In addition to setting up synthesis constraints, the user can also set up other timing constraints by using what is called a user constraints file (UCF). The UCF file for the design can be created by copying the **<design>.ncf** file that was created by the Synplify® tool to a UCF file called **<design>.ucf**. The UCF file can then be edited and changed to meet the timing requirements of the current design project. The UCF file will automatically be incorporated into the place-and-route process as long as the base part of the UCF filename matches the base part of the EDIF filename (e.g., pe0_design.ucf matches pe0_design.edf).

9.1.4.4 Running the Xilinx Place-and-Route Makefile

Once the Xilinx makefile has been edited and the timing constraints are completed, the design is ready to be placed and routed. Follow these simple steps to place-and-route a FPDP-E design:

1. If using an operating system other than Microsoft® Windows®, open a dos command prompt. If using an operating system other than Microsoft® Windows®, open a UNIX-like shell from which the **make** command will operate.
2. Change directories to the design project directory
3. Type **make** <design>**.hex**"

9.1.4.5 Analyzing Design Performance

Once the design has been placed and routed, the timing and area performance can be analyzed by referring to the following files.

<design>.par Area and timing results generated by the **par** tool
<design>.twr Timing report generated by the **trce** tool.

If the desired timing was not met, the designer can choose to modify the timing constraints or to modify the Xilinx® Alliance Series makefile options.

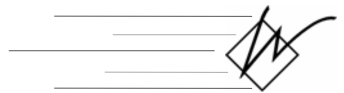
9.1.5 Transferring a PE design

The resulting binary file is used by the WILDSTAR™ or FIREBIRD™ API to load the Xilinx® image using the WS_ProgramPe function. On little-Endian machines (Intel), the binary file needs to be an .x86 file; on big-Endian machines (SUN, ALPHA, VxWorks), the binary file needs to be an .m68 file. This file is produced by the peutil program on Windows NT ®, and mcs2bin program on all other platforms.

If the VHDL development system and the host system are the same, mcs2bin and peutil can be run without transferring any files. If the host system is not the same as the VHDL development system and is not the same type of system (i.e., VHDL development performed on Intel, host system is SUN), the ASCII text <design>.mcs program must be transferred to the host system. The mcs2bin program then must be run on the host system.

To run the mcs2bin program under UNIX, execute "<ANNAPOLIS_BASE>**\host\tools\mcs2bin** <design>**.mcs"** for each design in the project.

INFORMATION NOTE
If the VHDL development system and the WILDSTAR™ host system are not of the same type, then the mcs2bi program must be run on the host system. Files produced by mcs2bin on a different type of system from the host system will not work on the host system.



INDEX

A

Active low signals, *1-4*
API, 1-3
Application Builder Online Help, 7-1
Application Programming Interface, 1-3

C

Cable, 4-10
Clock Configuration Register, **6-2**
Clocks, **6-1**

E

External I/O, 1-3

F

Features, *2-1*
FPDP, 1-3
FPDP Cable, 1-3
FPDP Cable Installation, 4-10
FPGA, 7-1
Frequency Parameters, **6-4**

G

General Specifications, **6-1**

H

Hardware, **6-1**
hardware installation, 4-1

I

I/O Card 0, 1-3
I/O Card 1, 1-3
Icons, 1-3
Input/Output, 1-3
Introduction, 7-1

K

KCLK, **6-3**

L

LED Definition, 3-4
Lower-Level Cores, 7-1

M

Main Board, 1-4
MCLK, **6-3**

P

PCLK, **6-3**
PE. See Processing Element
PMC, *8-50*
Processing Element, *1-4*

R

RACEway, *1-4*
Register Space Transactions, 8-18

T

Tools, 7-1
trace points, 7-1

U

UCLK, 6-3

V

VHDL Models, **8-1**

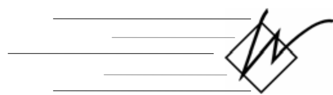
W

WILDSTAR™ Host Software, *1-4*
WILDSTAR™ II, **8-4**, **8-5**, **8-17**

X

XCLK, **6-3**

TM



ATTACHMENT A: GENERIC QUICK REFERENCE GUIDE

Clk_Std_IF Generics

The Clock Standard Interface has several generics that can be set by the user upon instantiation of this component. The table below indicates the generic name, type associated with the generic, valid values which can be used, and a description of each of the generic values:

Table A-1: Clk_Std_IF Generic Quick Reference Guide

Generic	Type	Value	Description
K_CLK_DLL_OUT M_CLK_DLL_OUT P_CLK_DLL_OUT U_CLK_DLL_OUT	Clk_DLL_Out_Type	USE_1X, USE_2X, BOTH	Select which output of the DLL is routed to the global buffer.
M_CLK_DLL_TYPE P_CLK_DLL_TYPE U_CLK_DLL_TYPE	Clk_DLL_Type	LOW_FREQ, HIGH_FREQ, NONE	Select which type of DLL to use.
M_CLK_DLL_DIVISOR U_CLK_0_DLL_DIVISOR U_CLK_1_DLL_DIVISOR	String	1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0	Sets the DV output of the specified DLL. Since these values are strings they must be enclosed in quotation marks.

LAD Mux Interface Generics

The LAD Mux interface has one generic that can be set by the user upon instantiation of this component. The table below indicates the generic name, type associated with the generic, valid values which can be used, and a description of each of the generic values:

Table A-2: LAD Mux Interface Quick Reference Guide

Generic	Type	Value	Description
K_Clk_x2	Boolean	TRUE/ FALSE	Set this value to TRUE when using a 66Mhz LAD Bus design. Set this value to FALSE when using a 33MHz LAD Bus design.

LAD_Mux_CPLD_IF Generics

The LAD Mux Clock PLD interface has several generics that can be set by the user upon instantiation of this component. The table below indicates the generic name, type associated with the generic, valid values that can be used, and a description of each of the generic values.

Table A-3: LAD_Mux_CPLD_IF Generic Quick Reference Guide

Generic	Type	Value	Description
K_Clk_x2	Boolean	TRUE/ FALSE	Set this value to TRUE when using a 66Mhz LAD Bus design. Set this value to FALSE when using a 33MHz LAD Bus design.
BASE	std_logic_vector(15 downto 0)	0x0000-0x7FFF	Sets the base address of the Clock PLD interface
INIT_EXT_CLK_SEL	std_logic	'0'or '1'	'0'External clock derived from BPE0 '1'External clock derived from BPE1
INIT_BRIDGE_CLK_SEL	std_logic_vector(1 downto 0)	0x0 – 0x3	(BPE0 uses Bit 0/BPE1 uses Bit 1) - 0 BPE U_CLk is PE's P_Clk - 1 BPE U_CLk is PE's U_Clk
INIT_PE_CLK_SEL	std_logic_vector(1 downto 0)	0x0 – 0x3	- Bit 0 0 P_Clk_0 is P_Clk 1 P_Clk_0 is U_Clk - Bit 1 0 P_Clk_1 is P_Clk 1 P_Clk_1 is IO_Conn(19)

LAD_Mux_BPLD_IF Generics

The LAD Mux Bridge PLD interface has several generics that can be set by the user upon instantiation of this component. The table below indicates the generic name, type associated with the generic, valid values that can be used, and a description of each of the generic values.

Table A-4: Bridge PLD Generic Quick Reference Guide

Generic	Type	Value	Description
K_Clk_x2	Boolean	TRUE/FALSE	Set this value to TRUE when using a 66Mhz LAD Bus design. Set this value to FALSE when using a 33MHz LAD Bus design.
Init_Data00	std_logic_vector(9 downto 0)	"0000000000" -> "1111111111"	Set the initial termination values for Port 0 Bank 0.
Init_Data01			Set the initial termination values for Port 0 Bank 1.
Init_Data10			Set the initial termination values for Port 1 Bank 0.
Init_Data11			Set the initial termination values for Port 1 Bank 1.
P0_25V_En	std_logic	'0', '1'	Set the initial value of the 2.5V enable for Port 0.
P1_25V_En			Set the initial value of the 2.5V enable for Port 1.

Mem36_Mux_IF Generics

The Mem36 Mux can be instantiated as a fair arbiter or a priority arbiter, each with its own set of generics. The tables below indicate the generic name, type associated with the generic, valid values which can be used, and a description of each of the generic values.

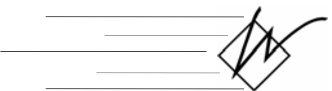
Table A-5: Mem36 Mux Priority Arbiter Interface Quick Reference Guide

Generic	Type	Value	Description
Num_Akk_FIFOs	Natural	Any natural number	
Avoid_Overflow	Boolean	TRUE/ FALSE	
Rotate_Priority			
Sticky_Priority			
Registered_Akks			
Registered_Reqs			
Register_Data			

Table A-6: Mem36 Mux Fair Arbiter Interface Quick Reference Guide

Generic	Type	Value	Description
Avoid_Overflow	Boolean	TRUE/ FALSE	
Register_Data			

TM



ATTACHMENT B: FPDP-E EXAMPLES

FPDP-E 2 FPFP-E Example

Figure B-1 is a block diagram describing the FPDP-E 2 FPDP-E Example supplied with the FPDP-E/Euro I/O card.

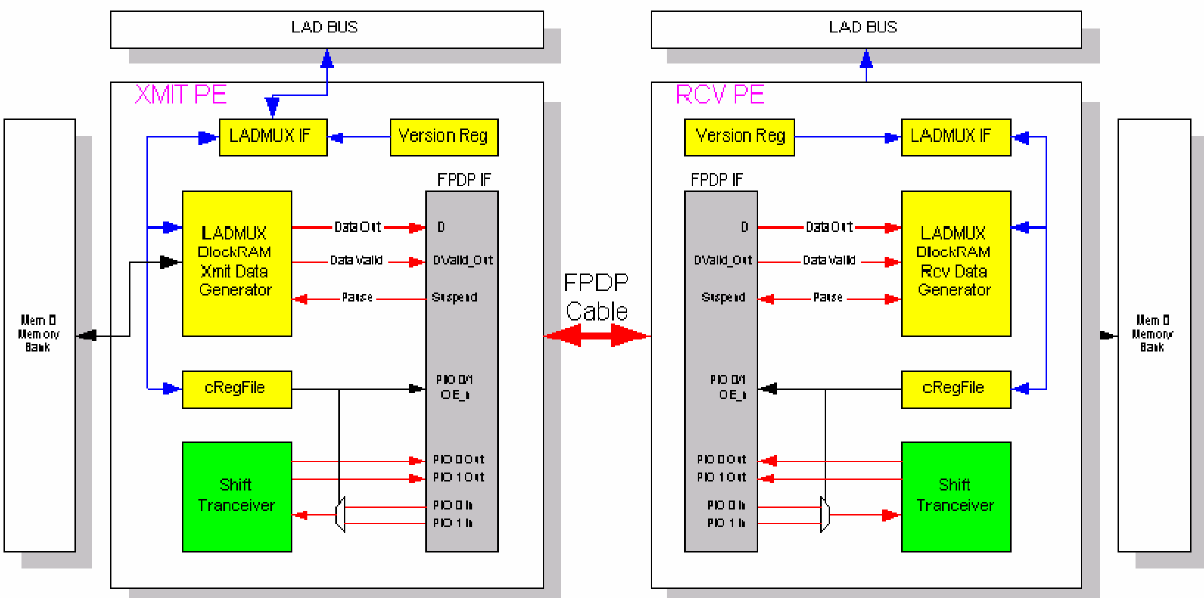


Figure B-1: FPDP-E to FPDP-E Example

This example demonstrates how to successfully transmit and receive data between two FPDP-E I/O cards. The transmit board is configured as the Transmit Master (T/M) and the receive board in configured as the Receive Master (R/M). The T/M supplies the clock to the R/M; however, the R/M is responsible for throttling the data between the two I/O cards. The “not ready” and the “synchronization” signals are not used in this example.

This fpdp2fpdp example uses one LAD Mux Mem36 Transmit Data Generator (LADMUX_Mem36_Xmit_DataGen) on the T/M board and one LAD Mux Mem36 Receive Data Generator (LADMUX_Mem36_Rcv_DataGen) on the R/M board. Each of the I/O cards also uses a serial shift transceiver that tests the programmable I/O bits. The PIO bits are controlled by the PLDs on the FPDP-E I/O cards and should operate independently of the data flow direction.

The host begins by resetting the PEs. Since this example tests both LVTTL and LVPECL FPDP transmit modes, the PLD must be initialized by the host before

any data is transferred. The SetPLD function call handles writing to both PEs and waiting for the transmit clock to be sourced on the receive board.

Once the receive board S_Clk is locked, the transmit and receive data generators are initialized. During initialization, a signature is read from the data generator telling the host what type of generator it is. All data generators are initialized with a header, footer, number of DWORDS to transfer, iteration count, and an "extra" register are written into each data generator. (For this example, the "extra" register is unused.)

If the data generator signature indicates it is a transmit data generator, the buffer containing the data to test will be written into its associated memory. If the data generator signature indicates it is a receive data generator, its associated memory values are set to 0x0. Once both data generators are initialized, a signal is sent to the transmit data generator and data begins flowing from the transmit data generator through the FPDP interface, out the FPDP cable to the receive FPDP interface, and finally through the receive data generator and into memory.

The suspend_n signal is used as a flow control between the two FPDP-E I/O cards, thus ensuring that data will not be dropped. The LADMux_Mem36_Xmit_Data_Generator consists of a LAD_Mux_Arb, a Mem36_Mux_Arb, a BlockRAM_FIFO, a LAD_Mux_cRegFile, a LAD_Mem36_Bridge and a small state machine. The LAD_Mux_Arb is used to take the incoming LAD_Mux_Vector and create three additional LAD_Mux_Vectors. The BlockRAM FIFO is used to cross clock domains between M_CLK and the receive clock (in this case they are the same). The LAD_Mux_cRegFile is used to store initialization values.

The state machine inside the LADMux_Mem36_Xmit_Data_Generator is responsible for sending header and footer information as well as for pulling data from the block RAM. Once the enable bit is set, the state machine starts by sending a header and setting the Data_Valid bit. Data is then pulled from the Block RAM and sent out the Data_Out port, Data_Valid will remain high. Once an internal counter reaches a value specified by the host, the state machine will send out the footer, indicating the cycle is complete, and Data_Valid will become low.

The LADMux_Mem36_Rcv_Data_Generators will continuously look for a header, once enabled. Once a header is received, data will be written to memory. Data will be written to memory until a footer is received, at which time Int_Req will be set high.

Since some platforms cannot detect an interrupt, the upper bit in the second control register will be set high. The host code in this example will ignore the interrupt and instead look for the upper-bit in the second control register to go high. Once the host reads a '1' in the second control register the host reads data

back from the LADMUX_Mem36_Rcv_Data Generator's memory port. After data is read, the values are compared for accuracy.

Once both the LVTTL and LVPECL modes are tested, a serial shift register, which can be configured as a transmitter or a receiver, is used to serially shift data between the two FPDP-E I/O cards using the PIO0 or PIO1 FPDP port signals. During this test, a pattern is written into the shift register configured as a transmitter and the same pattern in written into a compare register in the receive shift register.

Once the data is written, the shift register accepts data from one of the PIO signals and compares the serial data against the value written in the compare register. The host then sends a signal to the transmit shift register, which starts data flowing to the other FPDP-E I/O card. Once the receive shift register sees the pattern a bit is set high, and the host reads the value from the receive shift register. If the pattern matches, the example is successful. This is repeated twice, testing the PIO1 bit.

FPDP-E Memory Example

Figure B-2 is a block diagram describing the memory example supplied with the FPDP-E I/O card.

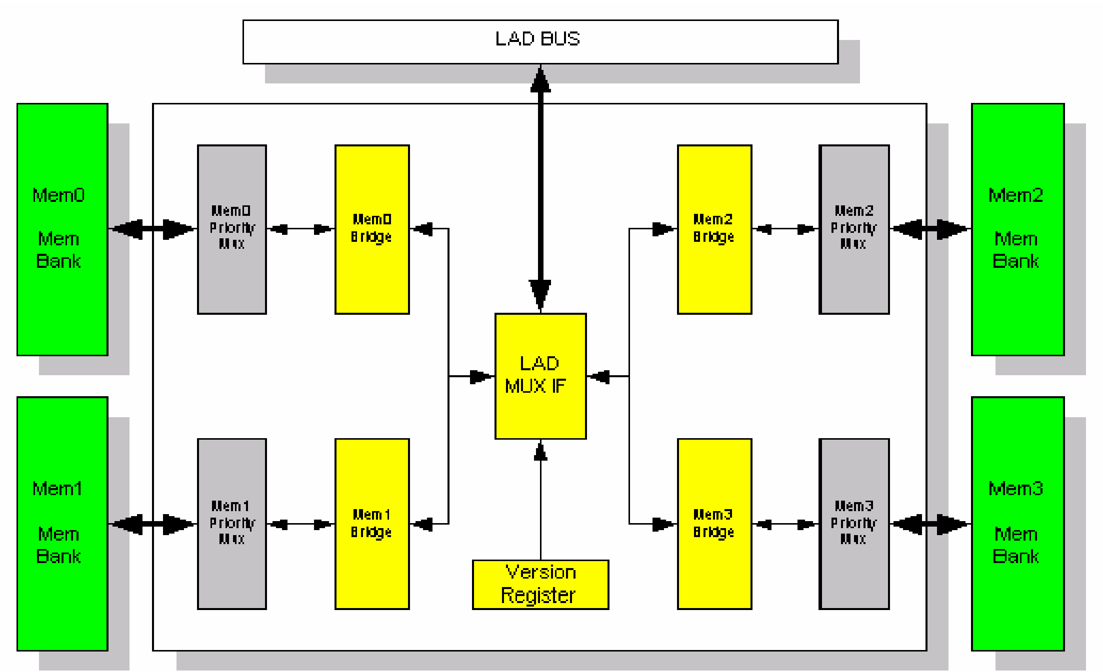


Figure B-2: Memory Example

The memory example uses a LAD_Mux_RegFile, four LAD_Mem36_Bridges and four memory priority mux interfaces to simply write and read data to and from the four local memory ports on the FPDP-E I/O card. Each shared memory

port uses a Mem36 Mux priority arbiter because only one client, a LAD_Mem36_Bridge, is used. The LAD_Mux_RegFile is used for version control.

The memory example generates test data and uses the WS_Mem_Write and WS_Mem_Read function calls, which are a set of WILDSTAR™ API calls created to communicate with the LAD Memory Bridges. The memory example tests each of the four memory ports with three types of data, a counter pattern to test the address lines, a "walking ones" pattern, and a random data pattern to test the data. The memory example writes data to and reads data from the memories, then checks for validity. If no errors are found, the test will report successful; otherwise, a list of errors will be displayed.

FPDP-E DMA Example

Figure B-3 is a block diagram describing the Direct Memory Access (DMA) example supplied with the FPDP-E I/O card.

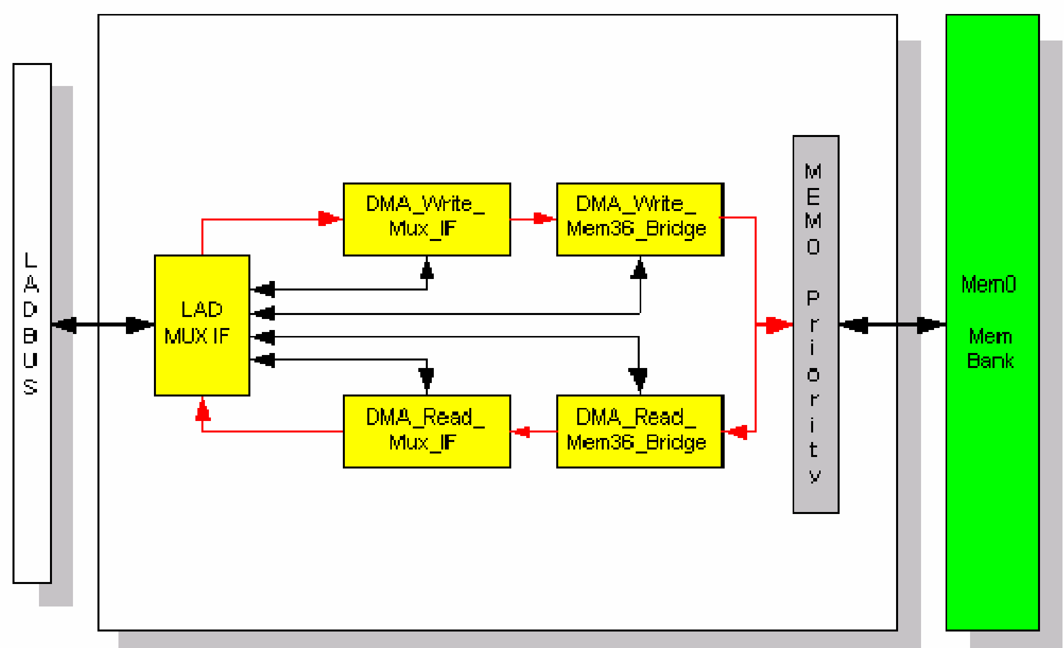


Figure B-3: DMA Example

The DMA example uses a DMA_Write_Mux_IF, a DMA_Read_Mux_IF, a DMA_Write_Mem36_Bridge and a DMA_Read_Mem36_Bridge to DMA data to and from memory port 0 on the FPDP-E I/O Card. Each of the DMA components is described in Chapter 8 of this manual.

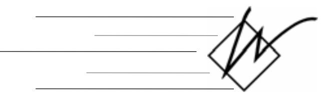
The DMA example generates test data on the host and uses programmed I/O writes to setup the DMA write bridge. In setting up the DMA_Write_Bridge, the starting address, and the number of DWORDS to transfer are stored into two

registers. When the write bridge is setup, the host issues a DMA write command, and data begins to be transferred to the PE. When data arrives at the DMA_Write_Mux_IF, the DMA_Bridge_Mem36_Bridge which is active (in this case there is only one bridge, but this doesn't always have to be the case), begins pulling data. The DMA_Write_Mux_IF controls the flow of the data, thus ensuring that no overflow occurs.

Once all of the data is transferred from the host, the DMA read process begins. The host sets up the DMA_Read_Mem36_Bridge by writing the memory starting address and the number of DWORDS to transfer. As soon as the number of DWORDS to transfer is not zero, the DMA_Read_Mem36_Bridge begins filling up the internal DMA_Read_Mux_IF's FIFO. After the host completes the programmed I/O write to set up the bridge, the host will issue a DMA Read command and wait for completion.

Once all data is transferred, the host compares the read and write buffer to check for errors. If no errors are found, the test will report successful.

TM



ATTACHMENT C: STANDARD INTERFACE COMPONENTS REPLACED BY MUX COMPONENTS (WILDSTAR™ AND FIREBIRD™ ONLY)

LAD Bus Standard Interface (LAD_Bus_Std_IF)

The local address data (LAD) bus is the primary means of communicating with the host system. The LAD bus standard interface (LAD_Bus_Std_IF) component provides the user interface to the LAD bus from within the PE device. The port signals of the LAD_Bus_Std_IF component are described below. The "User_In" and "User_Out" port signals should be used in the design to receive data from and send data to the LAD bus (and the host), respectively.

Table C-1: LAD Bus Standard Interface Component Port Signals

Signal Name	Width	Dir	Description
K_Clk	1	Input	VME/LAD bus clock signal; can be either 33 MHz or 66MHz
Global_Reset	1	Input	Global reset (or set) signal; usually connected to the GSR input of a STARTUP_VIRTEX component
Pads. Addr_Data	32	Bi-dir	Shared address/data bus pad signals; note that the address is a BYTE address
Pads.DS_n	1	Input	Data strobe pad signal; indicates when read data has been accepted by VME bus or when write data is valid on LAD bus
Pads.Reg_n	1	Input	Register select pad signal; indicates a register space access when '0' or memory space access when '1'
Pads.WR_n	1	Input	Write select pad signal; indicates a write cycle when '0' or read cycle when '1' (when Pads.CS_n is '0')
Pads.CS_n	1	Input	Chip select pad signal; indicates a valid LAD bus cycle when '0', otherwise no LAD bus cycle is taking place
Pads.DMA_Chan	2	Input	DMA channel number indicator; used to request status of one of four DMA channels
Pads.Int_Req_n	1	Output	Host interrupt request
Pads.DMA_Stat	2	Output	DMA status flag pad signals; indicates the status of the DMA channel that is currently selected by the DMA_Chan pad signal
User_In.Addr	23	Output	User interface to the LAD bus address; note that this is a DWORD address; also note that this address is automatically incremented each clock cycle during burst LAD bus cycles
User_In.Data_In	32	Output	User interface to the input data from the LAD bus
User_In.Reg_Strobe_n	1	Output	Register space access strobe signal; indicates a valid register space cycle when '0'
User_In.Mem_Strobe_n	1	Output	Memory space access strobe signal; indicates a valid memory space cycle when '0'
User_In.Write_Sel_n	1	Output	Write select interface signal; indicates a write cycle when '0' or a read cycle when '1' (when strobe is '0')
User_In.DMA_Chan	2	Output	User interface to the DMA channel number indicator; used to request status of one of four DMA channels
User_Out.Data_Out	32	Input	User interface to the output data to the LAD bus

Signal Name	Width	Dir	Description
User_Out.Int_Req_n	1	Input	User interface to host interrupt request line
User_Out.DMA_Stat	2	Input	User interface to the DMA status flags; indicates the status of the DMA channel that is currently selected by the DMA_Chan pad signal

On-board Memory Standard Interface (Mem_Std_IF)

The PE device has access to two identical 36-bit on-board synchronous SRAM memory ports that are referred to as memory ports 0-3. Each of these ports is accessed through a standard interface called the Mem_Std_IF. The port signals of the memory standard interface are shown in the table below. The "User_In" and "User_Out" port signals should be used in the design to receive data from and send data to the on-board memory devices, respectively.

Table C-2: Memory Standard Interface Component Port Signals

Signal Name	Width	Dir	Description
M_Clk	1	Input	Memory bus clock signal
Global_Reset	1	Input	Global reset (or set) signal; usually connected to the GSR input of a STARTUP_VIRTEX component
Mem_Pads.Data	36	Bi-dir	Data bus pad signals to/from memory
Mem_Pads.Addr	19	Output	Address bus pad signals
Mem_Pads.ACS_n	1	Output	Address chip select pad signal; used to switch between the two SRAM devices that make up the port; acts as an extra address bit
Mem_Pads.CS_n	1	Output	Chip select pad signal; enables both SRAM devices that make up the port; acts as an extra address bit
Mem_Pads.WE_n	1	Output	Write enable pad signal; indicates a write cycle when '0' or a read cycle when '1' (when Pads.CS_n is '0')
Mem_Pads.CKE_n	1	Output	Clock enable pad signal; enables the clocks on both SRAM devices that make up the port
Mem_Pads.Byte_WR_n	4	Output	Byte write enable signals, when low the corresponding byte can be written to memory
User_In.Data_In	36	Output	User interface to the data input from memory
User_In.Data_Valid_n	1	Output	User interface to the data input valid flag; when '0', indicates that the User_In.Data_In signal is valid
User_In.Byte_Sel_n	4	Output	User interface to the byte write enables, when each enable is '0' the corresponding byte can be written to memory.
User_Out.Addr	20	Input	User interface to the address to the Bridge PE or memory
User_Out.Data_Out	36	Input	User interface to the output data to the memory
User_Out.Strobe_n	1	Input	User interface to the memory access strobe; when '0', indicates a memory access is initiated
User_Out.Write_Sel_n	1	Input	User interface to the memory write select; indicates a write access when '0' or a read access when '1' (when User_Out.Strobe_n is '0')

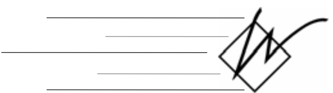
Clock PLD Interface (CSR_Std_If)

The Clock PLD standard interface (CSR_Std_IF) is a VHDL component that provides a user interface to the Clock PLD from within the FPDP PE device. The individual bit definitions for User_In.Data_In and User_Out.Data_Out are provided in Chapter 8. The port signals of the CSR_Std_IF component are described below. The “User_In” and “User_Out” port signals should be used in the design to receive data from and send data to the Clock PLD, respectively.

Table C-3: Clock PLD Interface Component Port Signals

Signal Name	Width	Dir	Description
Pads.Data	1	Bi-dir	Data pad to the FPDP clock PLD
Pads.Strobe_n	1	Output	Strobe_n pad to the FPDP clock PLD
Pads.WR_n	1	Output	WR_n pad to the FPDP clock PLD
Global_Reset	1	Input	Asynchronous reset signal for IOB registers
Clk	1	Input	Clock signal for IOB registers
User_In.Data_In	26	Output	Data read from PLD
User_In.Data_Valid_n	1	Output	User_In.Data_In is valid PLD data (read from PLD has completed)
User_Out.Data_Out	26	Input	Data to be loaded into PLD
User_Out.Load_n	9	Input	When this signal is a ‘1’ then values on User_Out.Data_Out will get loaded into PLD.

TM



ATTACHMENT D: FPDP-E I/O CARD DLL USAGE

The FPDP-E I/O card uses a Virtex-E™ part for the I/O PE. This part is equipped with eight delay lock loops (DLL) and four global buffers. Therefore, the Xilinx® place-and-route tools must be able to route four of the eight DLLl outputs (max) to each of the four global buffers. The additional four DLLs are called secondary DLLs. The arrangement of all the input clocks to the FPDP-E PE can be seen in Figure D-1.

When using these secondary DLLs, it will help the place-and-route tool if the DLL and all the related clock components are located in a user constraints file (ucf or ncf). For example, if Xilinx place-and-route is failing because it is unable to place the XLDCLK components, adding the following to the ncf will correct the problem:

```
inst
U_Clocks/G_XLD_OSC_SRC_G_XLD_OSC_SRC_U_XLD_CLK_DLL/G_L
F_DLL_G_LF_DLL_U_DLL LOC = DLL2S;

inst
U_Clocks/G_XLD_OSC_SRC_G_XLD_OSC_SRC_XLD_Clk LOC =
GCLKBUF2 ;
```

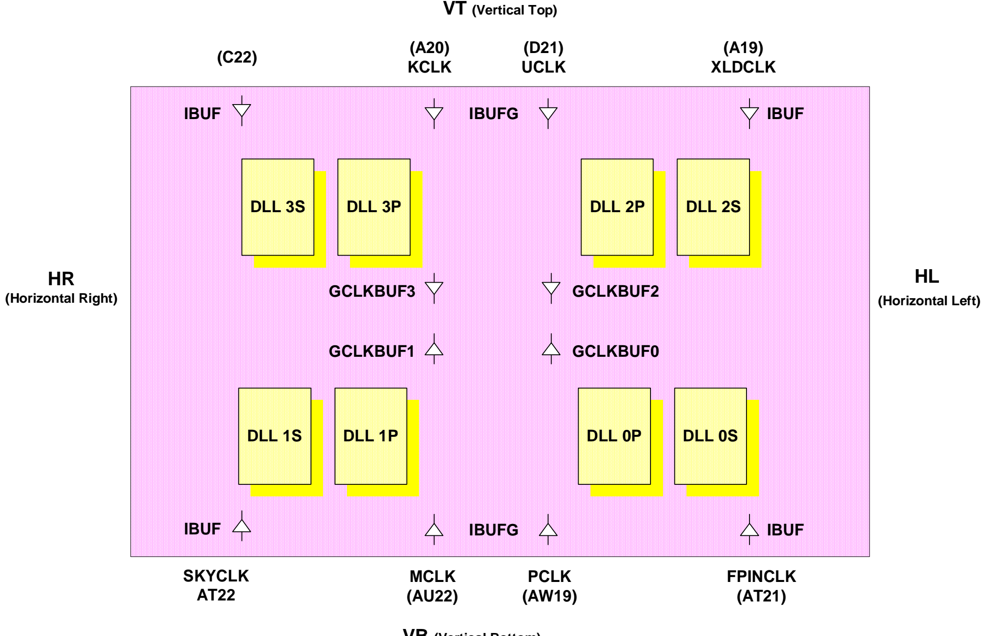


Figure D-1: FPDP-E I/O Card PE DLL Block Diagram

Additionally, when synchronizing MClk or UClk to another clock from across the chip, it may also be necessary to insert location constraints in order for Xilinx® place-and-route tools to place all of the components.