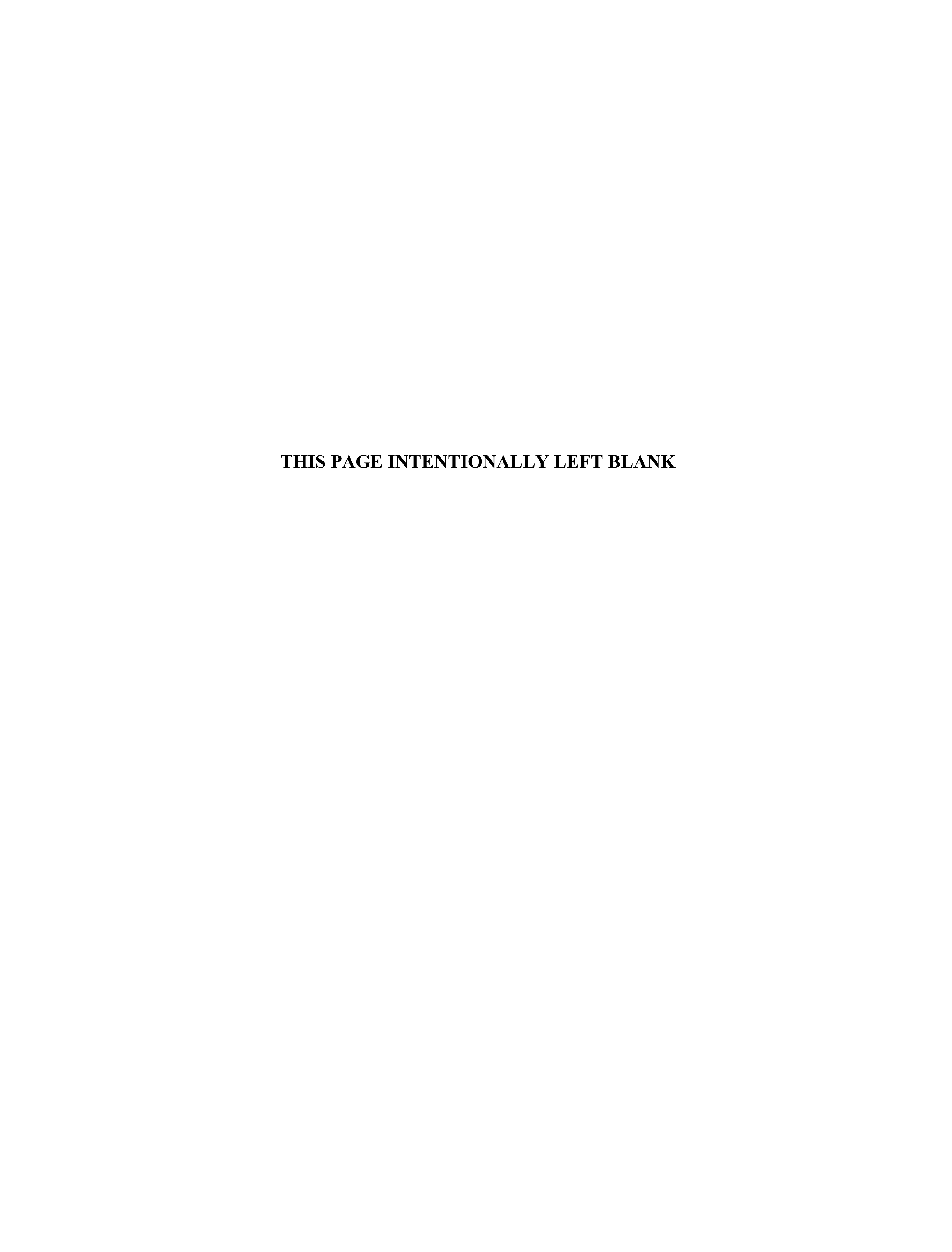
WILDSTAR™ ECL / PECL I/O Card Reference Manual

12934–0000 Revision 2.2

© Copyright 2002, 2003 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.


THIS PAGE INTENTIONALLY LEFT BLANK

ANNAPOLIS MICRO SYSTEMS, INC. - LICENSE AGREEMENT

WILDSTAR™ Family 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 U.S. 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 letters 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.

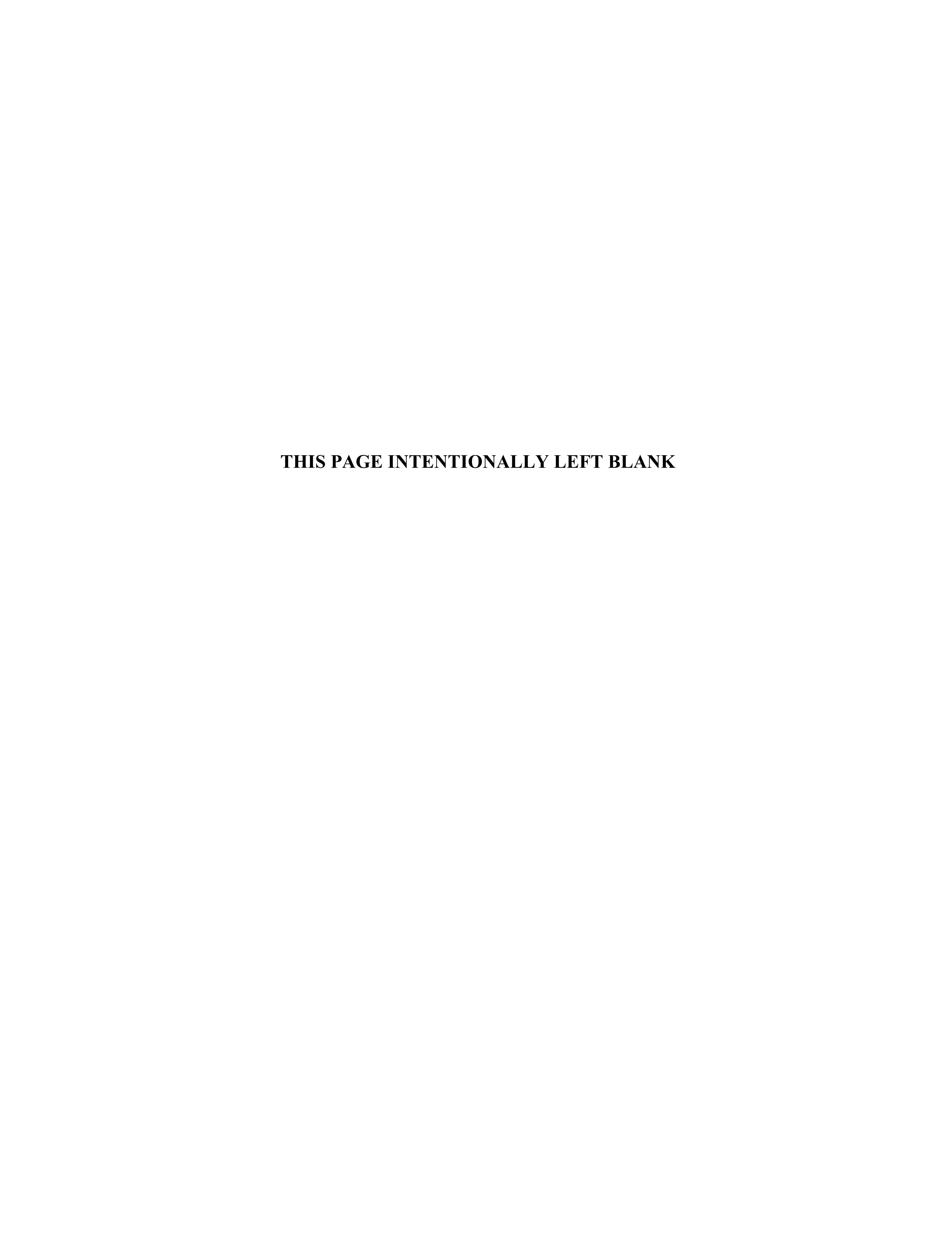

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

1. ABOUT THIS MANUAL 1-1

1.1 CHAPTER OVERVIEW 1-1
1.2 ICONS 1-2
1.3 KEY WORDS AND DEFINITIONS 1-3

2. WILDSTAR™ ECL/PECL I/O CARD 2-1

2.1 ECL/PECL I/O CARD ARCHITECTURE 2-4
2.1.1 I/O PE Interface to 68-Pin Connector 2-5
2.1.2 Card Interface with WILDSTAR™/VME and FIREBIRD™/PCI 2-5

3. GETTING STARTED 3-1

3.1 UNPACKING AND INSPECTING THE CARD 3-1
3.2 CARD COMPONENTS 3-2
3.3 FRONT PANEL—WILDSTAR™/VME MOTHERBOARD 3-4
3.4 ECL/PECL I/O BACK PLATE FOR FIREBIRD™/PCI MOTHERBOARD. 3-5
3.5 LED DISPLAYS ON THE WILDSTAR™ ECL/PECL I/O CARD 3-6
3.6 SWITCH SETTINGS FOR CARD VOLTAGE (ECL ONLY) 3-7

4. INSTALLING THE WILDSTAR™ ECL/PECL I/O CARD4-1

4.1 SYSTEM REQUIREMENTS 4-1
4.2 ECL/PECL I/O CARD COMPATIBILITY 4-1
4.3 HARDWARE INSTALLATION 4-2
4.3.1 Installing ECL/PECL on a WILDSTAR™/VME Board 4-2
4.3.2 Removing ECL/PECL from a WILDSTAR™/VME Board 4-5
4.3.3 Installing ECL/PECL on a WILDSTAR™-II /PCI Board 4-6
4.3.4 Removing ECL/PECL from a WILDSTAR™-II /PCI Board 4-8
4.3.5 Installing ECL/PECL on a FIREBIRD™/PCI Board 4-9
4.3.6 Removing ECL/PECL from a FIREBIRD™/PCI Board 4-10
4.4 CABLE REQUIREMENTS AND INSTALLATION 4-11

5. TECHNICAL SUPPORT 5-1

6. ECL/PECL I/O CARD HARDWARE REFERENCE 6-1

6.1 WILDSTAR™ ECL/PECL I/O CARD HARDWARE 6-1
6.2 GENERAL SPECIFICATIONS 6-1
6.3 POWER CONSUMPTION LIMITS 6-1
6.4 WILDSTAR™ ECL/PECL I/O CARD CLOCKS 6-2
6.4.1 Sourcing Motherboard Clocks (WILDSTAR™ and FIREBIRD™) 6-2
6.4.2 UCLK 6-2
6.4.3 MCLK 6-2
6.4.4 KCLK 6-3
6.4.5 XCLK 6-3

6.4.6 ECL Transmit and Receive Termination 6-3
6.4.7 PECL Transmit and Receive Termination 6-5
6.4.8 Frequency Parameters 6-7
6.4.9 Clock Skew 6-7

7. COREFIRE™ DESIGN SUITE SUPPORT 7-1

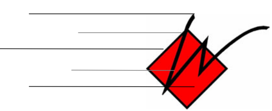
8. ECL/PECL I/O CARD VHDL MODELS REFERENCE .. 8-1

8.1 WILDSTAR™ ECL/PECL I/O CARD VHDL MODEL OVERVIEW 8-1
8.1.1 Block Diagrams 8-1
8.2 ECL/PECL I/O CARD PE MODEL 8-6
8.2.1 PE Pad Definitions and Locations 8-6
8.2.2 ECL/PECL I/O Card PE Interface Components 8-7
8.2.3 ECL/PECL I/O Card PE Components 8-8
8.3 ECL/PECL CABLES 8-23
8.3.1 ECL/PECL Cables 8-23
8.4 THE WILDSTAR™ FAMILY MUX LIBRARY 8-24
8.4.1 LAD_Mux Library 8-24
8.4.2 Mem_Mux Library 8-33
8.4.3 Programmed I/O Memory Bridge 8-37
8.4.4 DMA_Mux Library 8-40
8.4.5 DMA Memory Bridges 8-43
8.4.6 Host Model 8-48

9. ECL/PECL I/O CARD VHDL GUIDE 9-1

9.1 VHDL DESIGN CYCLE 9-1
9.1.1 ECL/PECL I/O Template VHDL Design Files 9-1
9.1.2 Simulating a VHDL Design 9-3
9.1.3 Synthesizing a VHDL Design 9-3
9.1.4 Place-and-Routing a Design 9-4
9.1.5 Transferring a PE design 9-6

INDEX........ I



1. ABOUT THIS MANUAL

This manual provides guidance for installing and using the WILDSTAR™ Emitter Coupled Logic (ECL) and Positive supply referred ECL (PECL) I/O Daughter cards, members of the WILDSTAR™ family of high-speed digital signal processing components. These cards combine high-speed interfaces with additional processing and memory bandwidth, resulting in a powerful and flexible tool with numerous computing applications. They are compatible with all WILDSTAR™/VME, -E/VME, WILDSTAR™-II /VME and /PCI, and FIREBIRD™/PCI and /cPCI motherboards.

The ECL/PECL I/O card is also fully compatible with the CoreFire™ Design Suite, an FPGA design application tool developed by Annapolis Micro Systems, Inc. As a dataflow-based FPGA design tool, CoreFire™ allows you to create designs in a fraction of the time required for a conventional VHDL-based control flow approach. CoreFire™ can be used to program ECL/PECL I/O card PEs and PEs on WILDSTAR™, WILDSTAR™-II, and FIREBIRD™ motherboards.

i

INFORMATION NOTE
In this document, **“ECL/PECL I/O card”** will stand collectively for both the WILDSTAR™ ECL and WILDSTAR™ PECL I/O Daughter cards, unless specifically noted otherwise.

i

INFORMATION NOTE
In this document, **“WILDSTAR™/VME”** will stand for WILDSTAR™/VME and WILDSTAR™-E/VME, unless otherwise indicated.

i

INFORMATION NOTE
In this document, **“FIREBIRD™”** will stand for FIREBIRD™/PCI and FIREBIRD™/cPCI, unless otherwise indicated.

1.1 Chapter Overview

- Chapter 1, “**About This Manual**,” outlines the conventions, icons, and key words used throughout the manual.
- Chapter 2, “**Introduction to the WILDSTAR™ ECL/PECL I/O Card**,” discusses its architecture and performance features.

- Chapter 3, "**Getting Started**," describes how to properly unpack and visually inspect the WILDSTAR™ ECL/PECL I/O card, including installation hardware, a license agreement, this reference manual, and a CD-ROM containing VHDL models and examples.

- Chapter 4, "**Installing the WILDSTAR™ ECL/PECL I/O Card**," explains how to properly install the card and the software included with it. Installation instructions are provided for the WILDSTAR™/VME, WILDSTAR™-II /VME and /PCI, and FIREBIRD™/PCI motherboards.

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

- Chapter 6, "**Hardware Reference**," includes general specifications, clock termination and sourcing options, along with other pertinent hardware information.

- Chapter 7, "**CoreFire™ Design Suite Support**," introduces the CoreFire™ Design Suite PE design application tool, describes its use with the I/O card, and summarizes the core libraries included with the tool.

- Chapter 8, "**VHDL Models Reference**," provides details of pad definitions and locations, VHDL interfaces to components of the ECL/PECL I/O card, and host models.

- Chapter 9, "**WILDSTAR™ ECL/PECL I/O Card VHDL Guide**," explains ways of using VHDL to program the ECL/PECL I/O card.

1.2 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.

1.3 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.

ECL

Emitter-Coupled Logic. Logic that switches current from one path to another.

External I/O

External Input/Output (I/O) for the WILDSTAR™ and FIREBIRD™ boards.

I/O Card 0

Designation of a WILDSTAR™ ECL/PECL I/O card installed in the top I/O card slot of the WILDSTAR™/VME board.

I/O Card 1

Designation of a WILDSTAR™ ECL/PECL I/O card installed in the bottom I/O card slot of the WILDSTAR™/VME board.

Motherboard

WILDSTAR™/VME, WILDSTAR™-II /VME and /PCI, or FIREBIRD™/PCI motherboard hosting the ECL/PECL I/O card.

_n

Signal name suffix denoting active low signal levels.

PE

Processing Element.

PECL

Positive Emitter Coupled Logic, a positive supply rail referred high-speed transmission standard.

Processing Element

A Xilinx® Virtex™ Field Programmable Gate Array (FPGA), comprising the main Processing Element on the ECL/PECL I/O card.

WILDSTAR™ Development Toolkit

Model technology, including ModelSim®, Synplicity® Synplify®, Xilinx® XACT*step*™, and Xilinx® Alliance™ series.

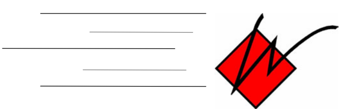
WILDSTAR™ Euro I/O Daughter Card
Eurocard form factor I/O daughter card for WILDSTAR™/VME.

WILDSTAR™ Host Software
Host software provided for the specified WILDSTAR™ board.

WILDSTAR™-II Host Software

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

WILDSTAR™ VHDL Models
Hardware models used for board-level VHDL simulation of applications.



2. WILDSTAR™ ECL/PECL I/O CARD

The WILDSTAR™ ECL/PECL I/O card provides high-speed interfaces with additional processing and memory bandwidth. It is capable of holding up to two million gates for each I/O PE on a motherboard, and up to eight megabytes of synchronous SRAM via four independent memory ports.

Other features of the ECL/PECL I/O card include:

- Motherboard with I/O card(s) installed occupies only a single VME or PCI slot.
- One dedicated Receive port with 15 ECL/PECL pairs (one of these pairs clocks the pin to the global buffer on the PE).
- One ECL port with 15 ECL/PECL pairs, factory-configured as Transmit or Receive.
- Rates of transmitting and receiving data up to approximately 300 Mbytes per second for each port.
- Ability to lock ECL/PECL I/O card and motherboard to external differential clock source (does *not* apply to WILDSTAR™-II motherboards).
- Full compatibility with the CoreFire™ Design Suite.

An I/O card placed in the first WILDSTAR™/VME daughter card slot is referred to as card "0." If an I/O card is placed in the second WILDSTAR™/VME daughter card slot, it is called card "1 (see Figure 2-1).

Since only one I/O daughter card slot can reside on a FIREBIRD™/PCI motherboard, the FPGA of the I/O card on this board is always referred to as "I/O card 0" (see Figure 2-2).

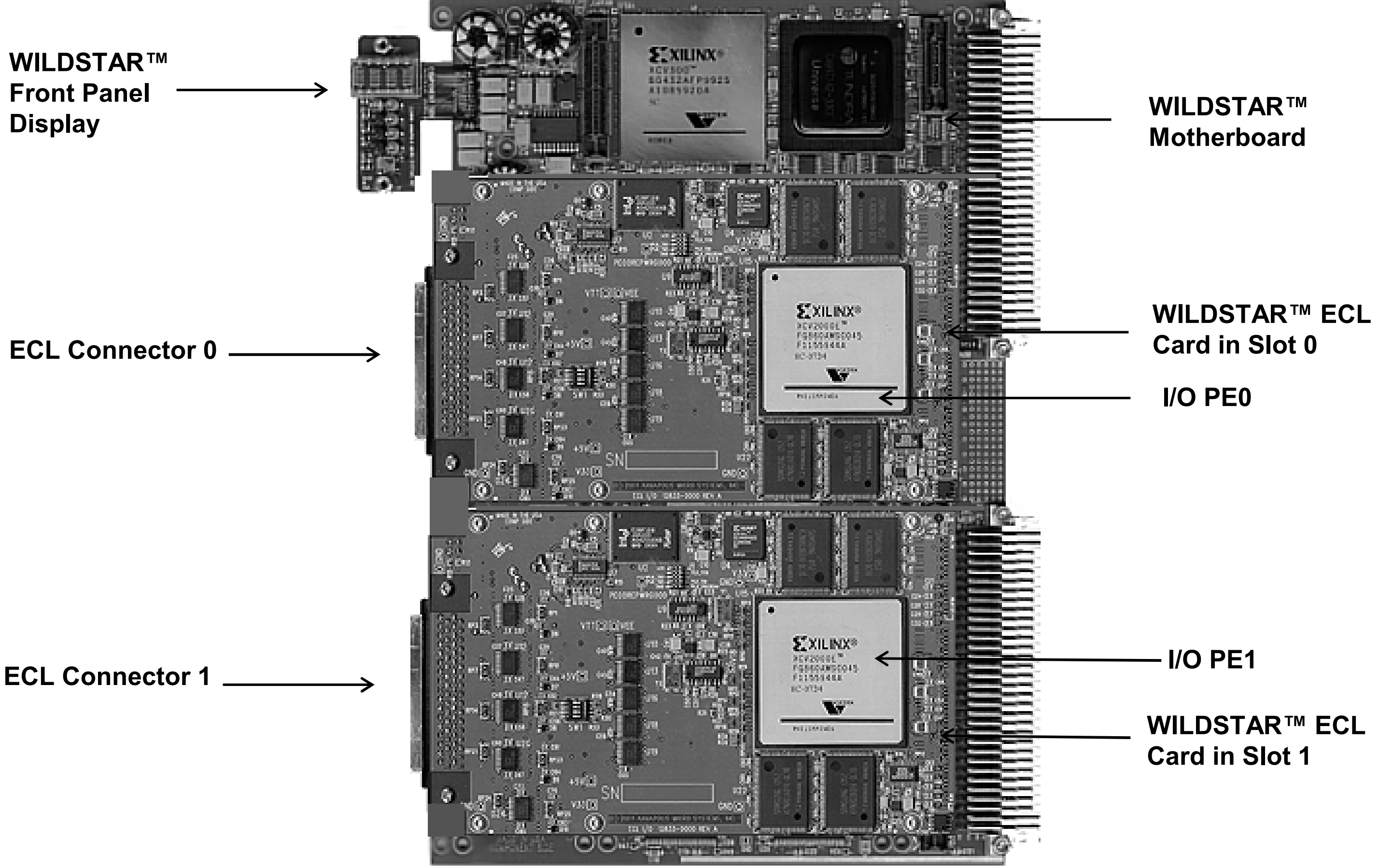


Figure 2-1: WILDSTAR™ Motherboard with WILDSTAR™ ECL/PECL Card 0 and WILDSTAR™ ECL/PECL Card 1

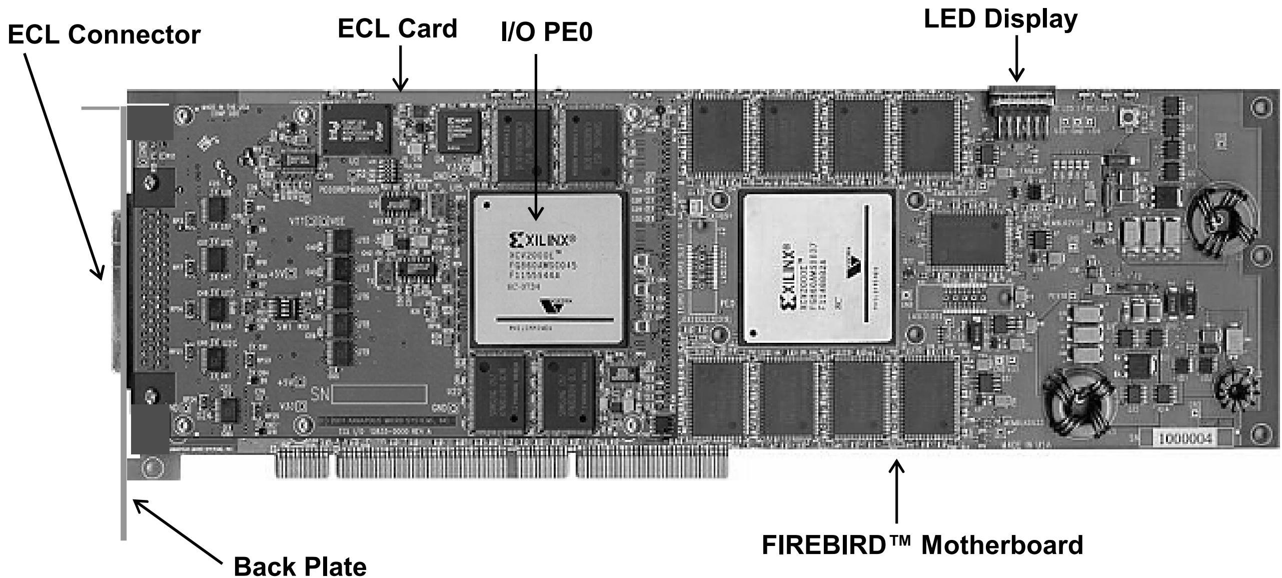


Figure 2-2: FIREBIRD™/PCI Motherboard with ECL/PECL I/O Card

2.1 ECL/PECL I/O Card Architecture

The ECL/PECL I/O card architecture is shown in Figure 2-3. The card communicates with the motherboard via an external I/O connector on the right side of the block diagram. The PE on the ECL/PECL I/O card is configured by the user.

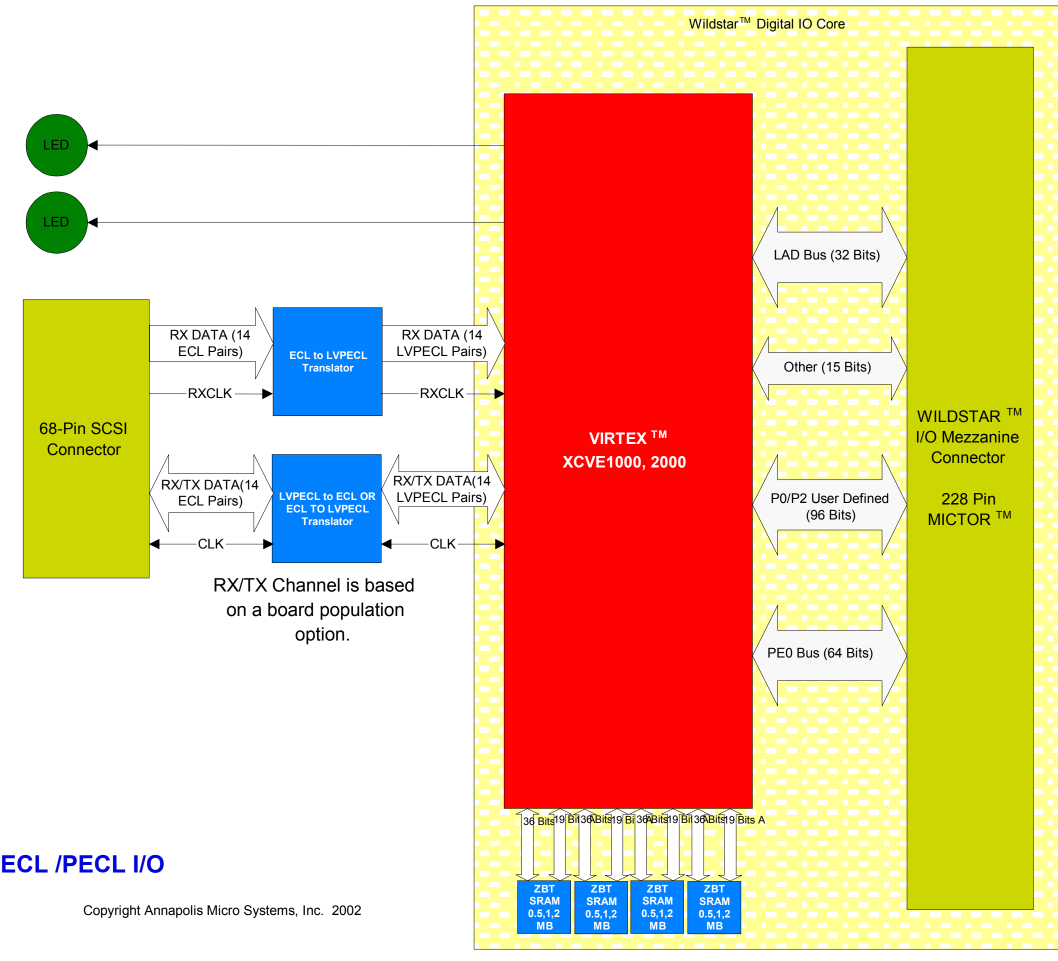


Figure 2-3: ECL/PECL I/O Card Block Diagram

2.1.1 I/O PE Interface to 68-Pin Connector

With a 15-bit wide differential Receive port, the interface between the I/O PE and the ECL/PECL I/O has been engineered for maximum design flexibility. One signal runs the clock and transmits to the global clock buffer. The second port, also a 15-bit wide differential port, is factory-configured to be either Receive or Transmit.

2.1.2 Card Interface with WILDSTAR™/VME and FIREBIRD™/PCI

Between PE0 on the WILDSTAR™/VME motherboard and each ECL/PECL I/O card are 66 connections. At 100 MHz, this results in a maximum throughput of 0.8 GBytes/sec for the WILDSTAR™/VME. In the case of the WILDSTAR™-E/VME, a speed of 133 MHz yields roughly 1.1 GBytes/sec of throughput for each ECL/PECL I/O card, with a dual card capacity of nearly 2.2 GBytes/sec.

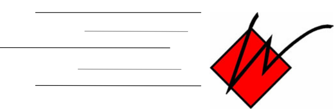
The FIREBIRD™/PCI motherboard and WILDSTAR™ ECL/PECL I/O card together have some 177 connections between them and a maximum speed of 150 MHz, resulting in a maximum throughput of over 3 GBytes/sec.

For WILDSTAR™-II /VME and /PCI boards, 168 pins allow for over 4 GBytes per second of throughput between the motherboard and the ECL/PECL I/O card.

In addition, the WILDSTAR™ ECL/PECL I/O card is compatible with RACEway™, Race++™, and Dual Race++™. RACEway™ and RACE++™ enhance VME bus bandwidth and allow point-to-point connections for transferring data between the host system and a WILDSTAR™/VME motherboard with an attached WILDSTAR™ ECL/PECL I/O card.

WILDSTAR™-II motherboards also support all RACE™ options, but they support none on the ECL/PECL I/O card.

THIS PAGE INTENTIONALLY LEFT BLANK



3. GETTING STARTED

3.1 Unpacking and Inspecting the Card

WILDSTAR™ ECL/PECL I/O cards are shipped in a static-sensitive pack. The card should remain sealed in this pack until installation time.

!

CAUTION

Before removing the board from the static pack, be sure to be grounded of all static electricity. Static discharge to the ECL/PECL I/O card could damage sensitive components.

Ensure that the following items are included with the card:

CD-ROMs

- WILDSTAR™ ECL/PECL I/O card VHDL CD, containing VHDL models, examples, and documentation

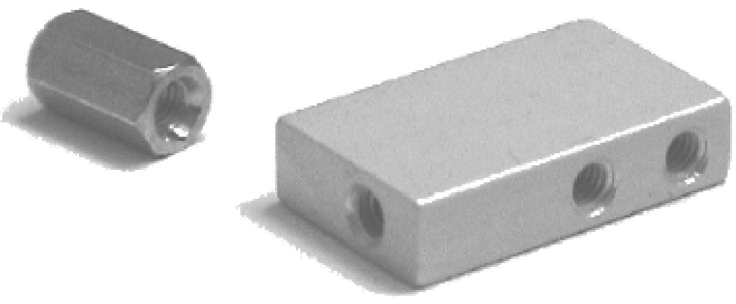
HARDWARE INCLUDED (illustrated below)

For WILDSTAR™/VME and WILDSTAR™-II /VME boards:

- 12 screws (only eight are used if motherboard has a memory card)
- 6 standoffs (only two are used if motherboard has a memory card)
- 4 memory mezzanine card spacers (only used with memory card)

For FIREBIRD™/PCI and WILDSTAR™-II /PCI boards:

- 16 screws (only 14 used with the FIREBIRD™ board)
- 4 standoffs
- Two mounting blocks
- Back plate



Standoff **Mounting Block**

When handling the ECL/PECL I/O 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 the card 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.

i

INFORMATION NOTE
Mounting blocks for use with the WILDSTAR™-II /PCI motherboard contain an extra screw hole for securing it to the board.

3.2 Card Components

Figures 3-1 and 3-2 illustrate major ECL/PECL I/O card part locations on component and solder sides. The WILDSTAR™/VME board front panel is shown in Figure 3-3, and the FIREBIRD™/PCI back plate is shown in Figure 3-4. Information on the location and significance of the WILDSTAR™ ECL/PECL I/O card LEDs is provided in Figure 3-5 and Table 3-1.

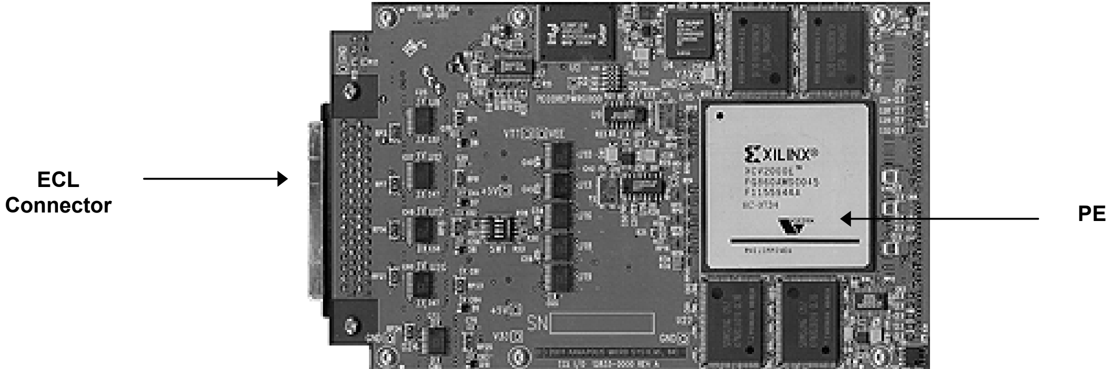


Figure 3-1: WILDSTAR™ ECL/PECL Card (Component Side)

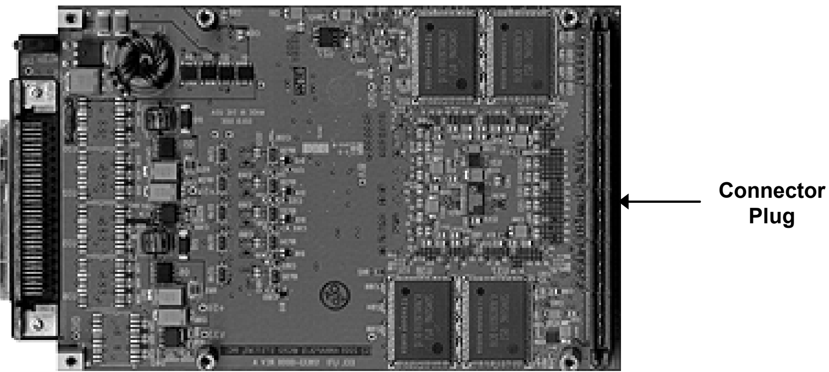


Figure 3-2: WILDSTAR™ ECL/PECL Card (Solder Side)

3.3 Front Panel—WILDSTAR™/VME Motherboard

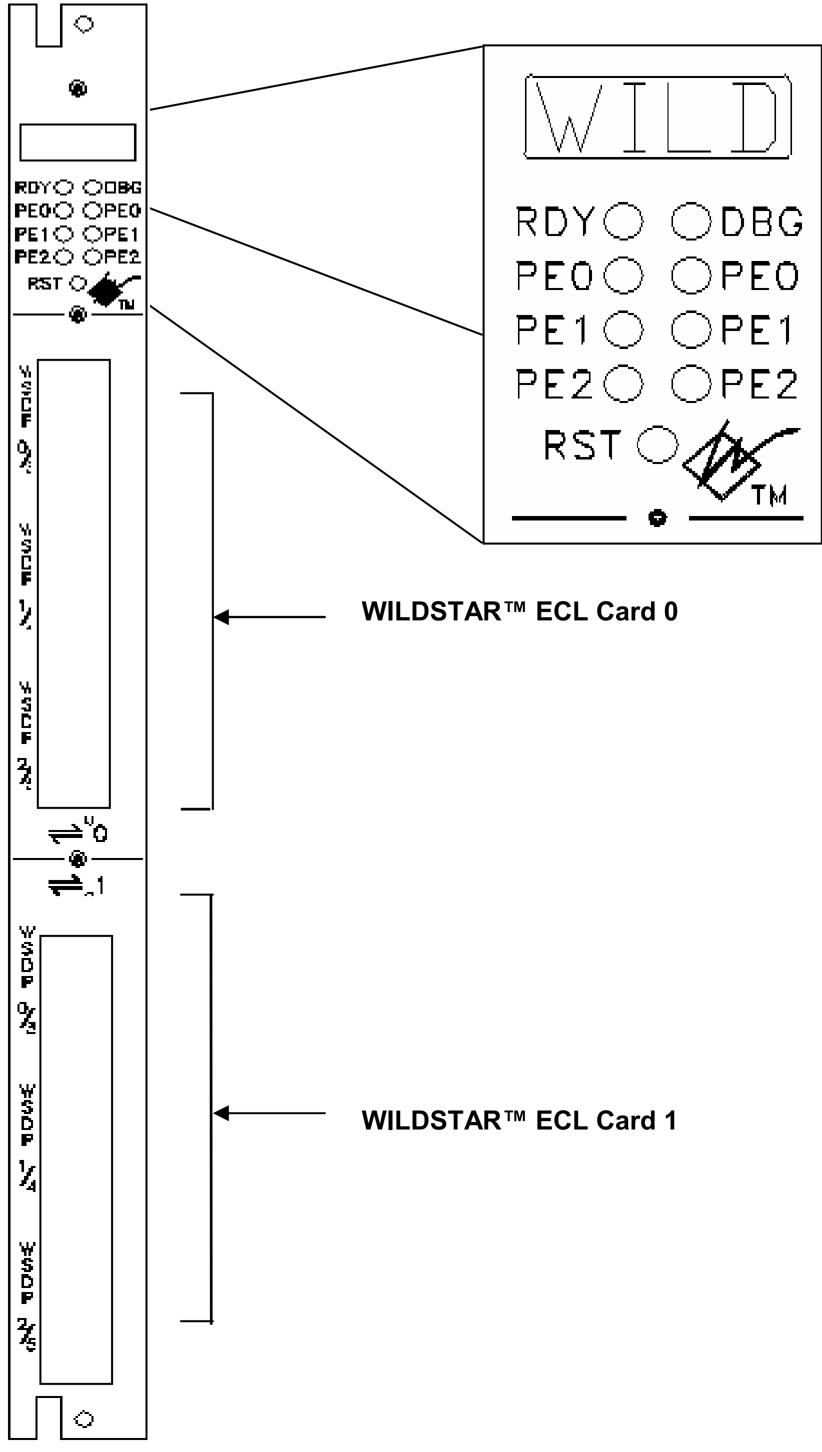


Figure 3-3: Front Panel for WILDSTAR™/VME with ECL/PECL Card(s)

3.4 ECL/PECL I/O Back Plate for FIREBIRD™/PCI Motherboard

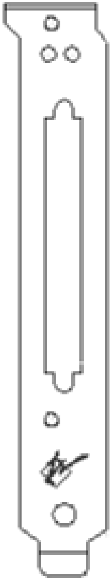


Figure 3-4: WILDSTAR™ ECL/PECL I/O Card Back Plate for FIREBIRD™/PCI

3.5 LED Displays on the WILDSTAR™ ECL/PECL I/O Card

A total of six LED functional and diagnostic displays reside on the ECL/PECL I/O card (Figure 3-5). If the ECL/PECL I/O card is used with an open chassis, the LEDs can provide useful diagnostic and performance information during application development.

Table 3-1 below lists the operations and POWER ON defaults of each LED display.

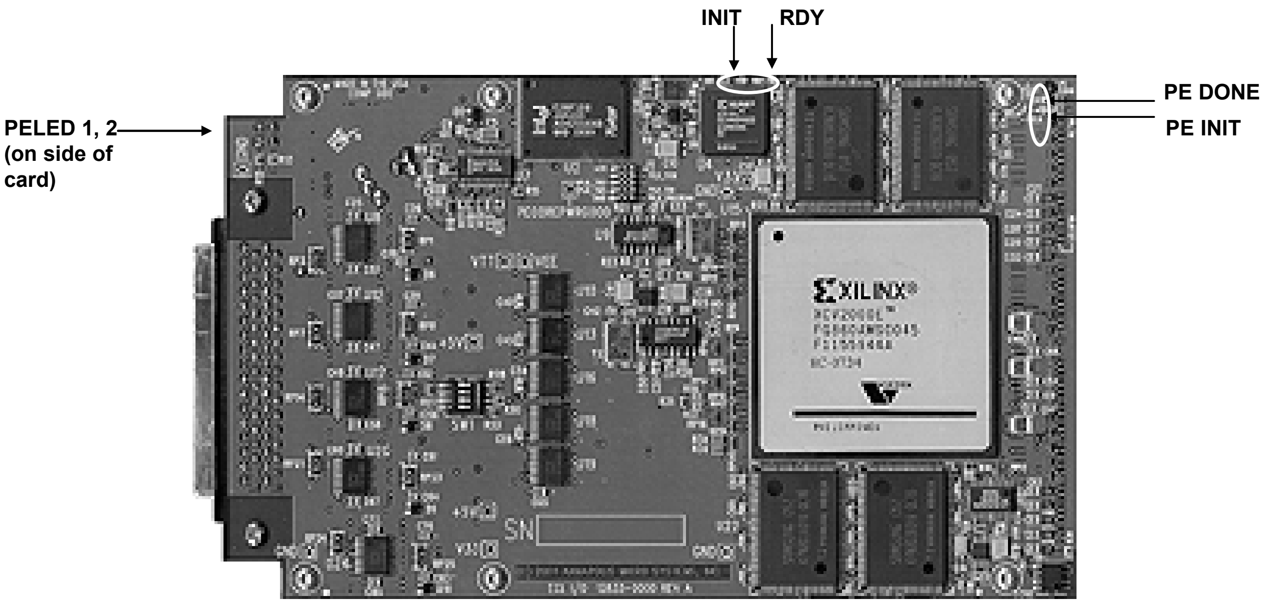


Figure 3-5: ECL/PECL Card Component Side Showing LEDs (1 – 6)

Table 3-1: ECL/PECL I/O Card LED Definitions

LED	LED Name	Board Label	LED Color	LED Default Power on State	LED Definition
LED 1	INIT	~INIT	Red	OFF	Reserved
LED 2	READY	RDY	Green	ON	Board Ready
LED 3	PE DONE	PE DONE	Green	OFF	PE Programmed
LED 4	~PE INIT	PE ~INIT	Red	OFF	PE Program Error
LED 5	PELED1	Same	Green	OFF	User LED
LED 6	PELED2	Same	Green	OFF	User LED

3.6 Switch Settings for Card Voltage (ECL Only)

On Dual Receive cards only, switch 4 determines whether the ECL I/O card will receive -5V ECL (off) or -3.3V ECL (on). On Transmit/Receive cards, this switch is inactive, since the voltage is factory preset to either 5 volts or -3.3 volts depending on customer preference.

Switches 3, 2, and 1 are "reserved."

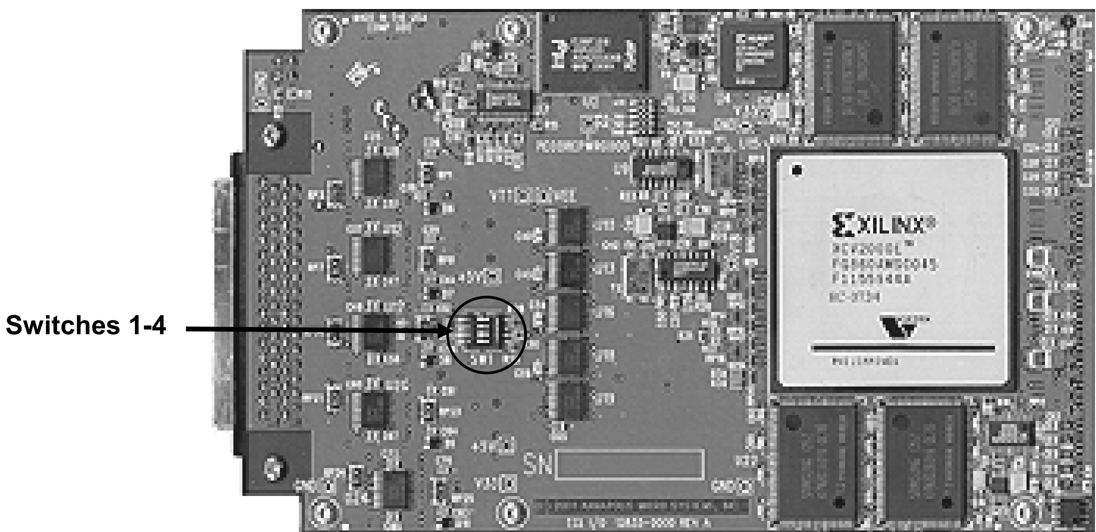
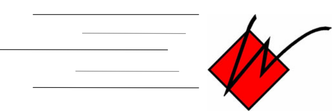


Figure 3-2: ECL I/O Card Switch Location

THIS PAGE INTENTIONALLY LEFT BLANK



4. INSTALLING THE WILDSTAR™ ECL/PECL I/O CARD

4.1 System Requirements

INFORMATION NOTE
In order to take advantage of all features of the WILDSTAR™ ECL/PECL I/O card, the computer host system should have the following: A WILDSTAR™/VME, WILDSTAR™-E/VME, or WILDSTAR™-II /VME or PCI motherboard; **OR** the FIREBIRD™/PCI motherboard. (See the appropriate hardware reference manual for system requirements).

4.2 ECL/PECL I/O Card Compatibility

CAUTION
On the WILDSTAR™/VME board, the determination of ECL/PECL card compatibility is based on the population of the +3.3V power supply. **If the optional +3.3V power supply is populated, an ECL card will not fit in the daughter card 0 slot (see Figure 4-1).** To check for the existence of the +3.3V power supply on the WILDSTAR™/VME board, also refer to Figure 4-1. This caution does *not* apply to the WILDSTAR™-II /VME motherboards.

INFORMATION NOTE
PECL can be populated without the I/O card power supply and use the WILDSTAR™-E/VME external I/O card supply.

Since the FIREBIRD™/PCI has the ability to "detect" and supply the required core voltage, I/O card compatibility is not an issue with this board.

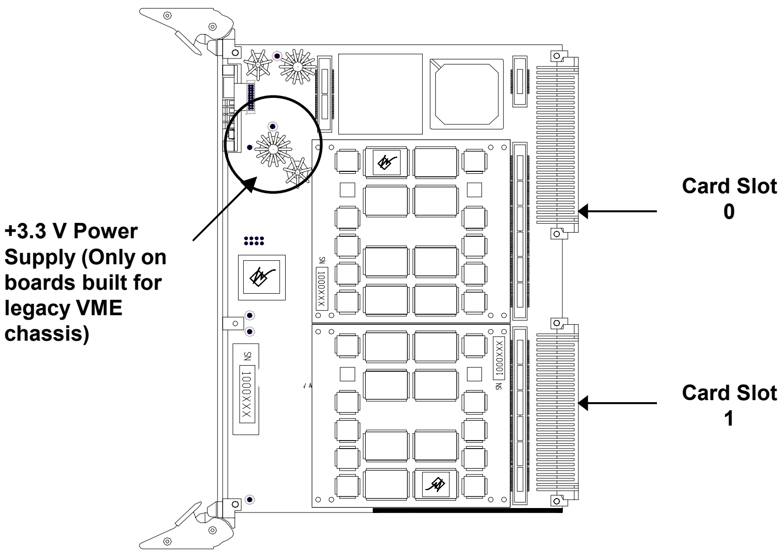


Figure 4-1: Location of +3.3V Power Supply on WILDSTAR™/VME Board Built for a Standard VME Chassis

4.3 Hardware Installation

This section provides diagrams and a series of steps for installing the ECL/PECL I/O card. Two sets of installation instructions are included here—one for WILDSTAR™/VME and one for FIREBIRD™/PCI.

4.3.1 Installing ECL/PECL on a WILDSTAR™/VME Board

To install one or more WILDSTAR™ ECL/PECL I/O cards on a WILDSTAR™/VME or WILDSTAR™-II /VME motherboard:

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 WILDSTAR™/VME or WILDSTAR™-II /VME motherboard.

4. Use the WILDSTAR™/VME ejectors to release the board from the slot (see Figure 4-2).
5. Remove the board from the chassis.
6. Remove the top, bottom, and middle screws securing the yellow front panel to the WILDSTAR™/VME motherboard. Next, remove two additional screws adjacent to the front panel ejectors (these are accessible on the solder side of the board). Once these screws are removed, *gently* pull the front panel away from the board. Be especially careful not to tug at the thin ribbon connecting the LED face to the board.
7. Place the WILDSTAR™/VME motherboard component-side up on a flat electrostatic-protected surface.
8. Place four mezzanine spacers over unpopulated screw holes atop the memory mezzanine card (see Figure 4-2 for location of memory cards and screw holes). Screws for these holes will go into the standoffs below the outer four corners of the memory card.

i

INFORMATION NOTE

If no memory card resides on the WILDSTAR™/VME, the ECL/PECL card is attached directly to the motherboard with six standoffs and twelve hex screws included with the ECL/PECL card. The standoffs are anchored with screws through the component side of the ECL/PECL card and the solder side of the motherboard. See Figure 4-3 for standoff locations.

Since **WILDSTAR™-II** motherboards have on-board memory, they do not support mezzanine memory cards. If you have a WILDSTAR™-II board, use standoffs and screws to attach the ECL/PECL I/O card directly to the board.

9. Remove the WILDSTAR™ ECL/PECL I/O card from its static-sensitive pack. Hold the card by its edges and place it gently on a flat, electrostatic-protected surface.
10. Attach two standoffs (included with the card package) to the solder side of the ECL/PECL card, using two of the hex screws provided to secure the standoffs.
11. Align the connector plug on the solder side of the ECL/PECL card with the connector receptacle on the WILDSTAR™/VME motherboard.
12. Secure the WILDSTAR™ ECL/PECL card by pressing gently above the connectors until the connectors are firmly seated.

13. Secure the card with four of the hex screws provided (see Figure 4-4). Turn the entire assembly over and use two more screws to secure the long standoffs to the motherboard. ***If there is no memory card on your WILDSTAR™/VME, see Step 8 above.***
14. Slide the ECL/PECL front panel over the connector. Replace the five screws removed in Step 6.
15. Reinstall the motherboard with the mounted WILDSTAR™ ECL/PECL card(s) into the computer. Refer to the installation section of the *WILDSTAR™ Reference Manual*, if necessary.

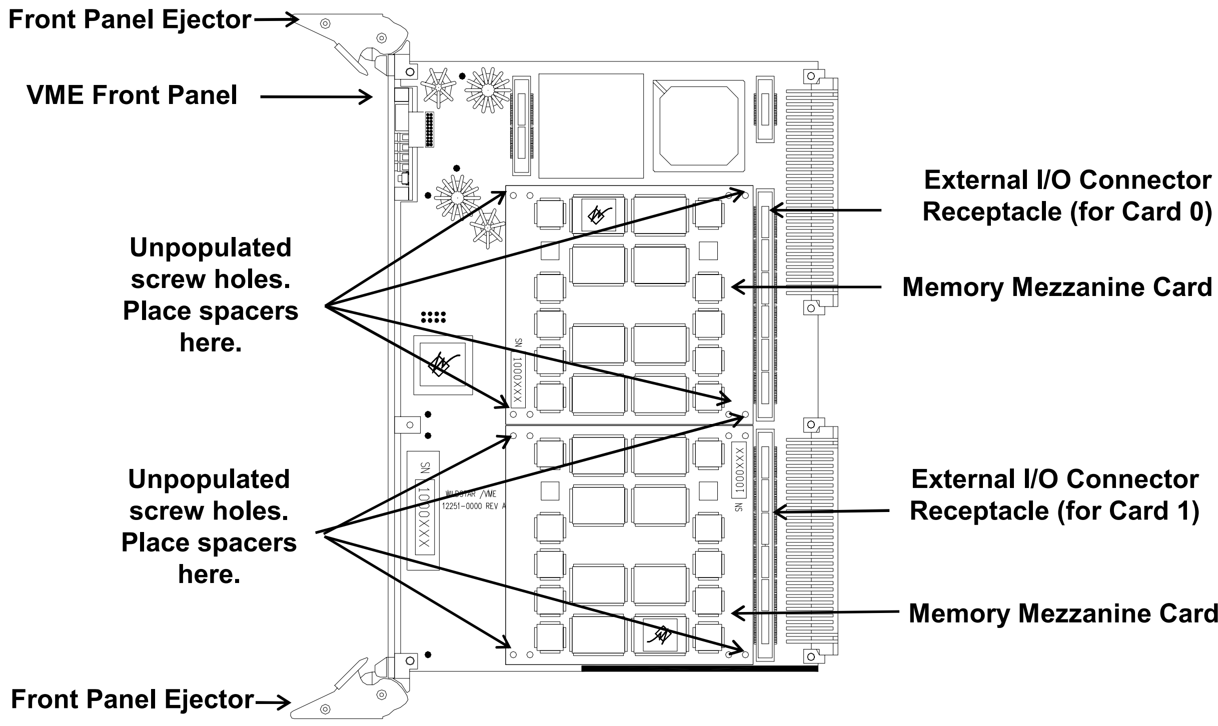


Figure 4-2: WILDSTAR™/VME Board with Memory Mezzanine Cards (Component Side)

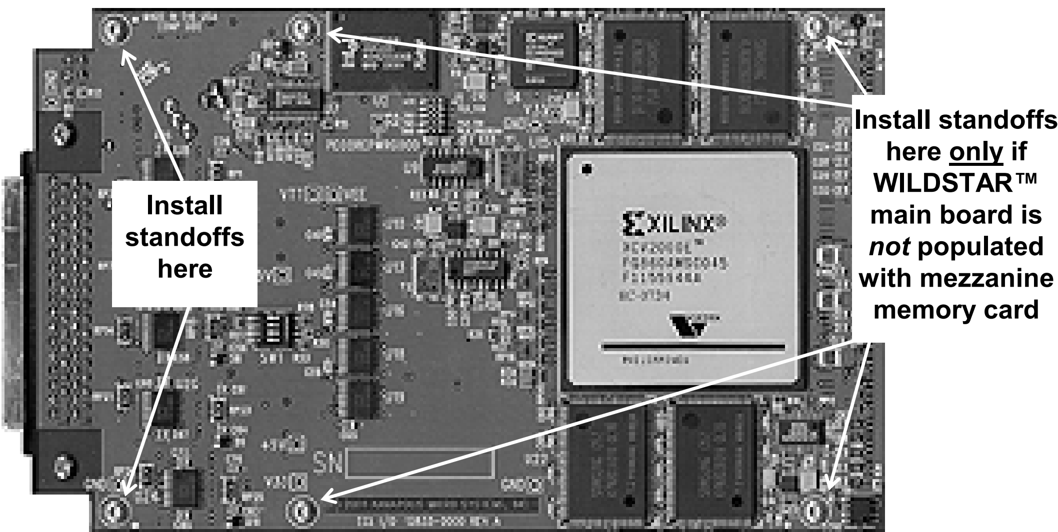


Figure 4-3: WILDSTAR™ ECL/PECL I/O Card (Solder Side) for WILDSTAR™/VME Motherboard Installation

4.3.2 Removing ECL/PECL from a WILDSTAR™/VME Board

To remove the WILDSTAR™ ECL/PECL I/O card from a WILDSTAR™/VME or WILDSTAR™-II /VME motherboard:

1. Connect the ground strap to the user.
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 WILDSTAR™/VME or WILDSTAR™-II /VME motherboard.
4. Remove any installed cables.
5. Use the WILDSTAR™/VME ejector handles to release the board from the slot (see Figure 4-2).
6. Remove the WILDSTAR™/VME motherboard from the chassis.
7. Remove the top, bottom, and middle screws securing the front panel to the WILDSTAR™/VME motherboard. Next, remove two additional screws adjacent to the front panel ejectors (these are accessible on the solder side of the board). Once these screws are removed, *gently* pull the front panel away from the board. Be especially careful of the thin ribbon connecting the LED face to the board.
8. Place the WILDSTAR™/VME motherboard component-side up on a flat electrostatic-protected surface.

9. Remove the six screws attaching the ECL/PECL I/O card to the WILDSTAR™/VME motherboard. Two of the screws (the ones holding the standoffs) will be accessible on the solder side of the motherboard.

INFORMATION NOTE
If no memory card resides on the WILDSTAR™/VME, the ECL/PECL I/O card is attached directly to the motherboard with six standoffs and twelve hex screws included with the ECL/PECL card. The standoffs are anchored with screws through the component side of the ECL/PECL card and the solder side of the motherboard. See Figure 4-3 for standoff locations. Since **WILDSTAR™-II** motherboards have on-board memory, they do not support mezzanine memory cards. If you have a WILDSTAR™-II board, use standoffs and screws to attach the ECL/PECL I/O card directly to the board.

10. Using a slight, gentle rocking motion, unseat the ECL/PECL I/O card connector plug from the WILDSTAR™/VME motherboard connector receptacle. Remove the card.
11. Remove the four gray mezzanine card spacers described in Section 4.3.1, Step 8, (*only* if a memory card resides on the motherboard).
12. Remove the two standoffs described in Section 4.3.1, Step 10.
13. Store the WILDSTAR™ ECL/PECL I/O card in a static-sensitive pack. Keep the spacers, standoffs, and screws for future use.
14. Replace the WILDSTAR™/VME front panel and secure with the five screws removed in Step 7.

To reinstall the WILDSTAR™/VME board, refer to the installation section of the *WILDSTAR™ Reference Manual*.

4.3.3 Installing ECL/PECL on a WILDSTAR™-II /PCI Board

[*NOTE: At this time, the WILDSTAR™-II /PCI motherboard only supports I/O cards in I/O Card Slot 0. Support of a second card will be available in later revisions*.]

To install the WILDSTAR™ ECL/PECL I/O card on a WILDSTAR™-II /PCI motherboard:

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 WILDSTAR™-II /PCI motherboard.
4. Remove the upward-facing screw securing the steel back plate of the WILDSTAR™-II /PCI 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 WILDSTAR™-II /PCI. Lay the back plate aside.
6. Place the WILDSTAR™-II /PCI component-side up on a flat electrostatic-protected surface.
7. Remove the WILDSTAR™ ECL/PECL card from its static-sensitive pack. Hold the card by its edges and place it carefully on a flat electrostatic protected surface.
8. Attach four standoffs (included with the card) to the component side of the WILDSTAR™-II /PCI motherboard by securing them with four 4mm hex screws (see Figure 4-4 for standoff locations). Screw them in from the solder side. *As noted earlier, the WILDSTAR™-II /PCI motherboard only supports I/O cards in I/O Card Slot 0 at this time.*

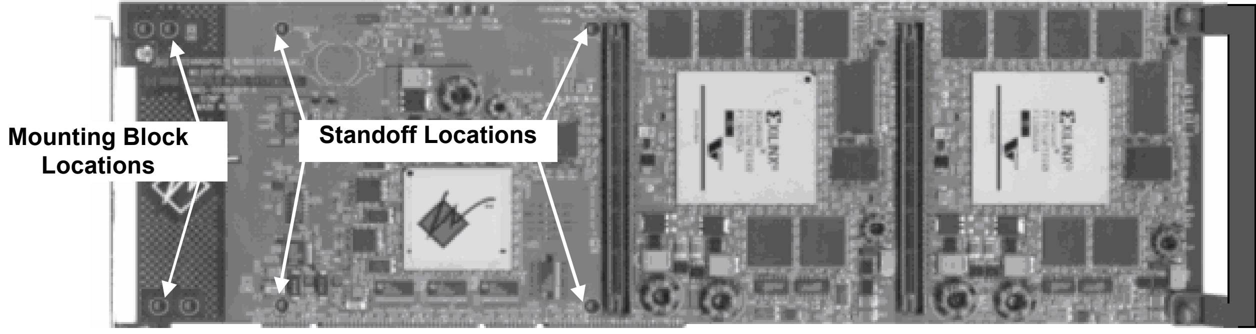


Figure 4-4: Standoff Locations for I/O Card Slot 0 of WILDSTAR™-II /PCI

9. Attach two mounting blocks to the component side of the WILDSTAR™-II /PCI motherboard by securing them with four hex screws through the motherboard's solder side (see Figure 4-4).
10. Align the connector plug on the solder side of the ECL/PECL I/O card with the I/O Card Slot 0 connector receptacle on the WILDSTAR™-II /PCI motherboard.

11. Press the card gently against the connector receptacle until the connector is completely seated.
12. Install six hex screws (packaged with the ECL/PECL I/O card assembly) through the component side of the 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 your ECL/PECL I/O card and slide it over the ECL/PECL connector. Attach it to the mounting blocks using two of the screws provided.

Reinstall the motherboard with the mounted ECL/PECL I/O card into the computer. Refer to the installation section of the *WILDSTAR™-II Reference Manual*, if necessary.

4.3.4 Removing ECL/PECL from a WILDSTAR™-II /PCI Board

To remove the WILDSTAR™ ECL/PECL I/O card from a WILDSTAR™-II /PCI motherboard:

1. Connect the ground strap to the user and power off.
2. If necessary, remove the cover to the system chassis in order to gain access to the WILDSTAR™-II /PCI motherboard.
3. Remove any installed cable by gently removing it from the ECL/PECL connector.
4. Remove the upward-facing screw securing the steel back plate of the WILDSTAR™-II /PCI 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 WILDSTAR™-II /PCI. 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 WILDSTAR™ ECL/PECL I/O card to the WILDSTAR™-II /PCI motherboard.
8. Using a slight and gentle rocking motion, unseat the WILDSTAR™ ECL/PECL I/O card connector plug from the motherboard connector receptacle. Remove the card.
9. Remove the four standoffs (described in Section 4.3.3, Step 8) from the WILDSTAR™-II /PCI motherboard.
10. Also, remove the two mounting blocks (described in Section 4.3.3, Step 9) from the motherboard.

11. After removing the card, store it in a static-sensitive pack. Keep the installation hardware for future use.
12. Replace the standard back plate to the WILDSTAR™-II /PCI and secure with two screws.

4.3.5 Installing ECL/PECL on a FIREBIRD™/PCI Board

To install the WILDSTAR™ ECL/PECL I/O card on a FIREBIRD™/PCI motherboard:

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 FIREBIRD™/PCI motherboard.
4. Remove the upward-facing screw securing the steel back plate of the FIREBIRD™/PCI 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 FIREBIRD™/PCI. Lay the back plate aside.
6. Place the FIREBIRD™/PCI component-side up on a flat electrostatic-protected surface.
7. Remove the ECL/PECL I/O card from its static sensitive pack. Hold the card by its edges and place it carefully on a flat electrostatic protected surface.
8. Attach four standoffs (included with the card) to the component side of the FIREBIRD™/PCI motherboard by securing them with four 4mm hex screws (see Figure 4-5 for standoff locations). Screw them in from the solder side.
9. Attach two mounting blocks to the component side of the FIREBIRD™/PCI motherboard by securing them with two hex screws through the motherboard's solder side (see Figure 4-5).
10. Align the connector plug on the solder side of the ECL/PECL I/O card with the connector receptacle on the FIREBIRD™/PCI motherboard.
11. Press the card gently against the connector receptacle until the connector is completely seated.

12. Install six hex screws (packaged with the ECL/PECL I/O card assembly) through the component side of the 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 your ECL/PECL I/O card and slide it over the ECL/PECL connector. Attach it to the mounting blocks using two of the screws provided.

Reinstall the motherboard with the mounted ECL/PECL I/O card into the computer. Refer to the installation section of the *FIREBIRD ™ Reference Manual*, if necessary.

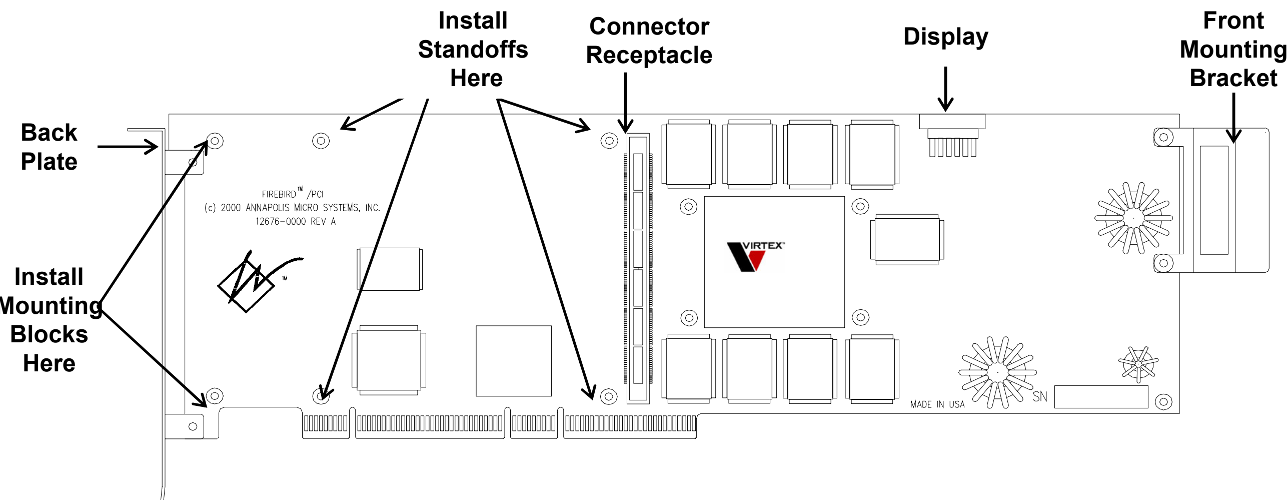


Figure 4-5: FIREBIRD™/PCI Board (Component Side)

4.3.6 Removing ECL/PECL from a FIREBIRD™/PCI Board

To remove the WILDSTAR™ ECL/PECL I/O card from a FIREBIRD™/PCI motherboard:

1. Connect the ground strap to the user and power off.
2. If necessary, remove the cover to the system chassis in order to gain access to the FIREBIRD™/PCI motherboard.
3. Remove any installed cable by gently removing it from the ECL/PECL connector.
4. Remove the upward-facing screw securing the steel back plate of the FIREBIRD™/PCI 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 FIREBIRD™/PCI. 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 WILDSTAR™ ECL/PECL I/O card to the FIREBIRD™/PCI motherboard.
8. Using a slight and gentle rocking motion, unseat the WILDSTAR™ ECL/PECL I/O card connector plug from the motherboard connector receptacle. Remove the card.
9. Remove the four standoffs (described in Section 4.3.5, Step 8) from the FIREBIRD™/PCI motherboard.
10. Also, remove the two mounting blocks (described in Section 4.3.5, Step 9) from the motherboard.
11. After removing the card, store it in a static-sensitive pack. Keep the installation hardware for future use.
12. Replace the standard back plate to the FIREBIRD™/PCI and secure with two screws.

To reinstall the FIREBIRD™/PCI motherboard, refer to the installation section of the *FIREBIRD™ Reference Manual*.

4.4 Cable Requirements and Installation

For proper connection of the ECL/PECL I/O card, a 68-pin twisted-pair ribbon cable (not included with card) is recommended. This cable must have a 68-pin SCSI connector (AMP Part No. 750913-7) to be compatible with the card.

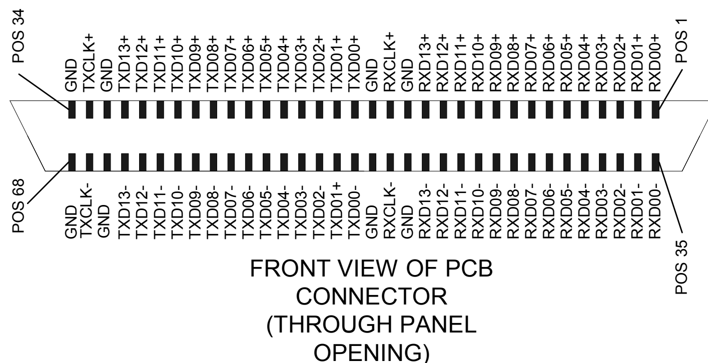
Each ECL/PECL I/O card has two 14-bit differential ECL/PECL ports that can be connected to an ECL/PECL port on the same card, on a second ECL/PECL I/O card on the same board, or between ECL/PECL I/O cards on a second board. In order to simulate an ECL/PECL cable, the ECL/PECL I/O VHDL model includes a cable package.

Figure 4-6: ECL/PECL I/O Card Ribbon Cable

Figure 4-7: ECL/PECL I/O Card Pigtail Cable

Pinout functions for the two card connector options are displayed below. The two factory-configured options are Transmit/Receive and Dual Receive:

RECEIVE DATA PORT (RXD[13:00]) ALWAYS CONFIGURED TO RECEIVE DATA



CONNECTOR IS AN AMP 787169-7 (JACKSCREW RETENTION), OR 787349-7 (LATCH RETENTION) 68-PIN CONNECTOR.

Figure 4-8: ECL/PECL I/O Connector Pinout Functions, Transmit/Receive Card

RECEIVE DATA PORTS (RXD1 and RXD2 [13:00]) ALWAYS CONFIGURED TO RECEIVE DATA

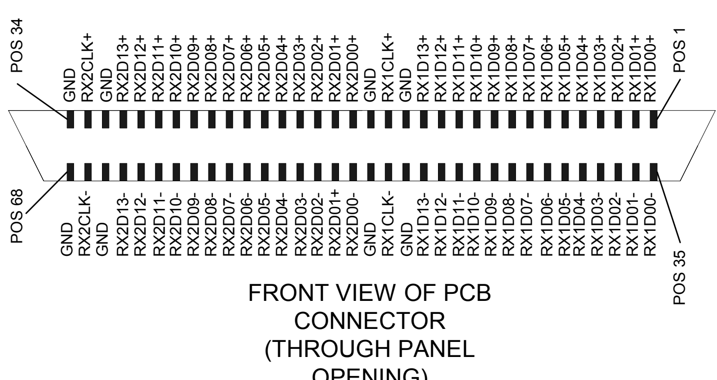
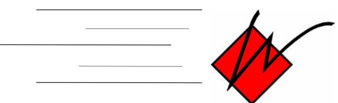


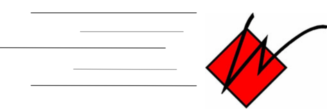
Figure 4-9: ECL/PECL I/O Connector Pinout Functions, Dual Receive Card



5. TECHNICAL SUPPORT

If you have any questions about installing, programming, using, or maintaining the WILDSTAR™ ECL/PECL I/O card, please call the WILDSTAR™ Technical Support staff at (410) 841-2514, fax at (410) 841-2518, or send e-mail to wftech@annapmicro.com.

THIS PAGE INTENTIONALLY LEFT BLANK



6. ECL/PECL I/O CARD HARDWARE REFERENCE

6.1 WILDSTAR™ ECL/PECL I/O Card Hardware

This chapter contains hardware reference information, including clocking, switching, termination options, and electrical specifications, for the WILDSTAR™ ECL and PECL I/O cards.

6.2 General Specifications

Table 6-1 specifies the physical dimensions and operating range for the WILDSTAR™ ECL/PECL I/O card:

Table 6-1: WILDSTAR™ ECL/PECL I/O Card Specifications

WILDSTAR™ ECL/PECL I/O Card for WILDSTAR™/VME and FIREBIRD™/PCI	
Physical Dimensions:	Length: 144.8mm/5.84 in Width: 91.44mm/3.6 in Thickness: 1.4mm/.055 in
Operating Range:	Temperature: 0º to 70ºC

6.3 Power Consumption Limits

!

CAUTION

According to WILDSTAR™/VME specifications, the total power should not exceed **35** watts per slot. According to VME64X specifications, the total power per slot should not exceed 45 watts. The user must ensure that the base system is capable of supplying adequate power and cooling to run the application.

!

CAUTION

According to FIREBIRD™/PCI specifications, the total power should not exceed **25** watts per slot. If the estimated power consumption for an application exceeds **25** watts per slot, the user must ensure that the base system is capable of supplying adequate power and cooling to run the application.

i

INFORMATION NOTE

On Dual Receive cards only, switch 4 determines whether the card will receive -5V ECL (off) or -3.3V ECL (on). Transmit/Receive cards will always be set to receive -5V ECL.

6.4 WILDSTAR™ ECL/PECL I/O Card Clocks

The WILDSTAR™ ECL/PECL I/O card has numerous clocks that can be used by the PE, including MCLK, UCLK, KCLK, XCLK (RACEway™ clock in Figure 6-1). The motherboard sources MCLK, UCLK, and KCLK. The P2 connector on the VME backplane line sources XCLK on WILDSTAR™/VME. One of the dedicated Receive channel signals is connected to a global clock buffer

In addition to the four clock lines sourced by the motherboard, UCLK_SRC and MCLK_SRC are sourced by the WILDSTAR™ ECL/PECL I/O card to the motherboard. These signals allow for MCLK and/or UCLK to be synchronized to an external clock.

6.4.1 Sourcing Motherboard Clocks (WILDSTAR™ and FIREBIRD™)

The WILDSTAR™ ECL/PECL I/O card provides the ability to synchronize an external clock to one of the motherboard clocks. MCLK, UCLK, or both can be synchronized from the front panel clock or from one of the ---- channels on WILDSTAR™ ECL/PECL I/O card. MCLK and UCLK are distributed on the board with very low skew.

The WILDSTAR™ API allows the source of MCLK, UCLK, or both to be external. Refer to the *WILDSTAR™ Reference Manual* for API details.

6.4.2 UCLK

UCLK is a user configurable clock. It is asynchronous to MCLK and KCLK. The source of UCLK is selectable via the WILDSTAR™ host software between the WILDSTAR™ programmable oscillator and the external I/O cards.

6.4.3 MCLK

All PE memory is run exclusively from MCLK. MCLK is used to derive PCLK on the WILDSTAR™/VME and FIREBIRD™/PCI motherboards, and is asynchronous to UCLK and KCLK. The source of MCLK is selectable via the WILDSTAR™ host software between the WILDSTAR™ programmable oscillator and the external I/O card.

6.4.4 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.

6.4.5 XCLK

XCLK comes from the VME backplane on WILDSTAR™/VME. When a WILDSTAR™ ECL/PECL I/O card is plugged into 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.

6.4.6 ECL Transmit and Receive Termination

Transmit and Receive termination configurations vary between the ECL and PECL I/O cards. Figure 6-1 and 6-2 display the Transmit and Receive terminations of the ECL I/O card. Two voltage options are available for Transmit termination on the ECL card (-5 Volt and -3.3 Volt). Transmit and Receive electrical specifications are described in Tables 6-2 and 6-3.

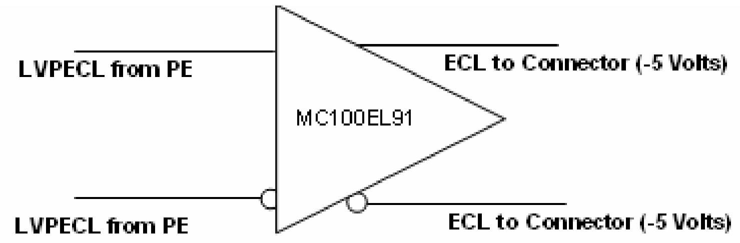


Figure 6-1: ECL I/O Card Transmit Termination (with -5 Volt option)

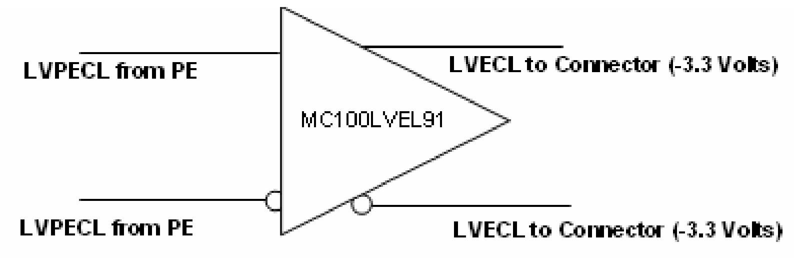


Figure 6-2: ECL I/O Card Transmit Termination (with -3.3 Volt option)

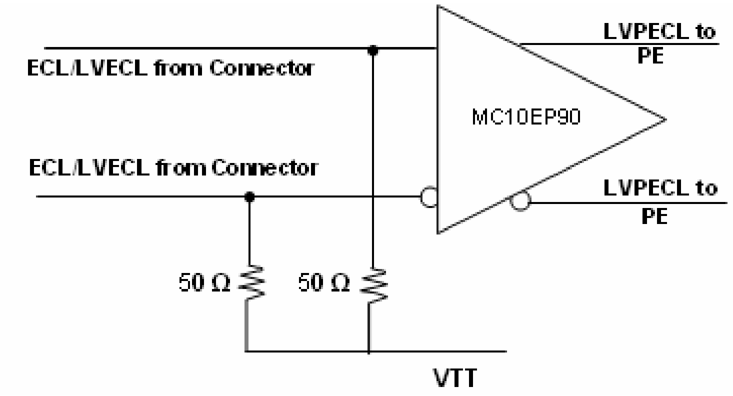


Figure 6-3: ECL I/O Card Receive Termination

Table 6-2: Receive Electrical Specifications for ECL

25°C

V_{IH} =	-1145 to -820 mV
V_{IL} =	-1870 to -1545 mV
V_{IHCMR}	-3.0 to 0.0V for V_{EE} = -5V
	-1.3 to 0.0V for V_{EE} = -3.3V
V_{PP}	Min: 150mV
	Nominal: 800mV
	Max: 1200mV

Table 6-3: Transmit Electrical Specifications for ECL

25°C

V_{OH} =	Min: -1025mV
	Nom: -955mV
	Max: -880mV
V_{OL} =	Min:-1810mV
	Nom:-1705mV
	Max: -1620mV

Table 6-4: Transmit Electrical Specifications for -3.3V Transmit/Receive ECL

	25°C
V_{OH}	Min: –1025mV
	Nom: –955mV
	Max: –880mV
V_{OL}	Min: –1810mV
	Nom: –1705mV
	Max: –1620mV

6.4.7 PECL Transmit and Receive Termination

The following diagrams show the Transmit and Receive terminations of the PECL I/O card. There are two voltage options available for Receive termination on this card—Parallel and VTT. Transmit and Receive electrical specifications are described in Tables 6-5 and 6-6.

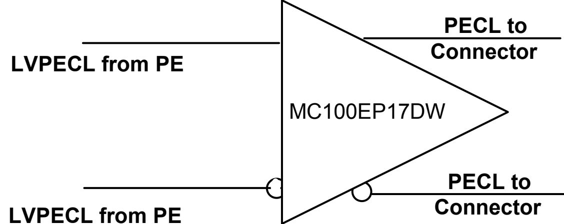


Figure 6-4: PECL Transmit Termination

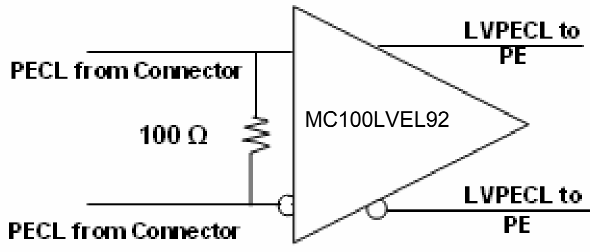


Figure 6-5: PECL Receive Termination (Parallel Option)

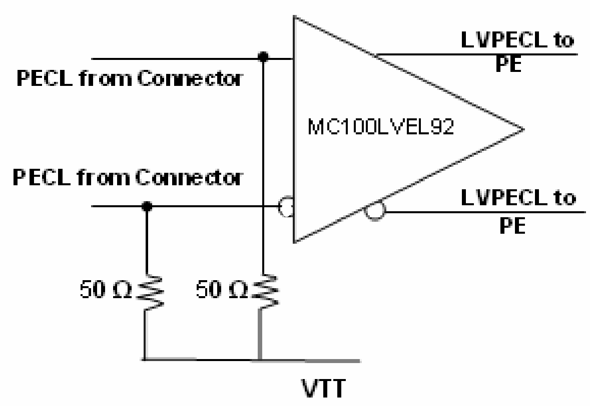


Figure 6-6: PECL Receive Termination (VTT Option)

Table 6-5: Receive Electrical Specifications for PECL

V_{IH} =	3835 to 4120 mV
V_{IL} =	3190 to 3515 mV
V_{IHCMR} V_{PP}<500mV: V_{PP}>=500mV:	1.2 to 4.8V 1.4 to 4.8V
V_{PP}	150 to 1000mV

Table 6-6: Transmit Electrical Specifications for PECL

25°C

V_{OH} =	Min: 3930mV
	Nom: 4055mV
	Max: 4180mV
V_{OL} =	Min:3130mV
	Nom:3255mV
	Max:3380 mV

6.4.8 Frequency Parameters

Table 6-7:2 describes each clock, its frequency ranges, and its destination on the ECL/PECL I/O card. The clock source can differ depending on the motherboard that carries the card.

Table 6-7: ECL/PECL I/O Card Clock Description

Clock	Full Name	Clock Source	Frequency Range	Destination
UCLK	User Clock	WILDSTAR™/VME	10 MHz to 100 MHz	Optional on I/O PE
		WILDSTAR™-E/VME	10 MHz to 133 MHz	
		FIREBIRD™	10 MHz to 133 MHz	
MCLK	Memory Clock	WILDSTAR™/VME	*15 MHz to 100 MHz	PE and Memories
		WILDSTAR™-E/VME	*15 MHz to 100 MHz	
		FIREBIRD™	10 MHz to 133 MHz	
KCLK	System Clock	WILDSTAR™/VME WILDSTAR™-E/VME FIREBIRD™	33 MHz or 66 MHz	PE
XCLK	VME Backplane Generated Clock	WILDSTAR™/VME WILDSTAR™-E/VME	0 MHz to 100 MHz	Optional on PE
MCLK_SRC	External User Clock Source	PE	**7.5 MHz to 50 MHz	WILDSTAR™/VME or FIREBIRD™/PCI
UCLK_SRC	External User Clock Source	PE	7.5 MHz to 50 MHz	WILDSTAR™/VME or FIREBIRD™/PCI

* When using memory cards, the minimum is 25 MHz.

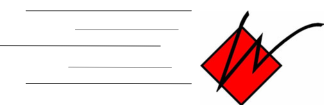
** When using memory cards, the minimum is 12.5 MHz.

6.4.9 Clock Skew

Table 6-8: ECL/PECL I/O Card Clock Skew

Symbol	Description		Typical	Maximum	Units
T_{SKEW}	Maximum Clock Skew (UCLK, MCLK, KCLK)	Single Board	TBD	1.5	ns

THIS PAGE INTENTIONALLY LEFT BLANK



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.

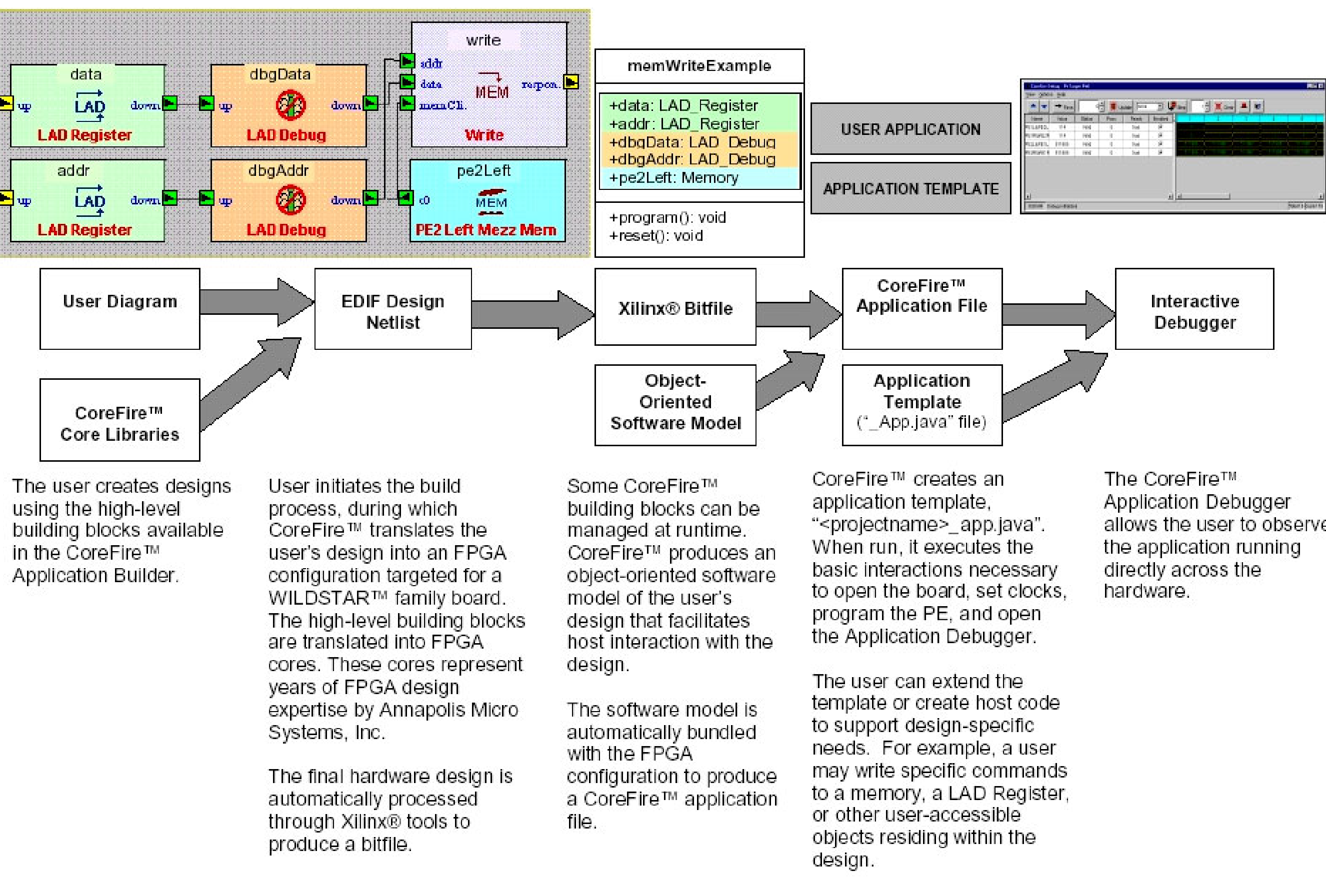


Figure 7-1: CoreFire Design Flow Model

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. You can also build customized cores, called Core Macros, using cores from the libraries (excluding board-specific libraries).

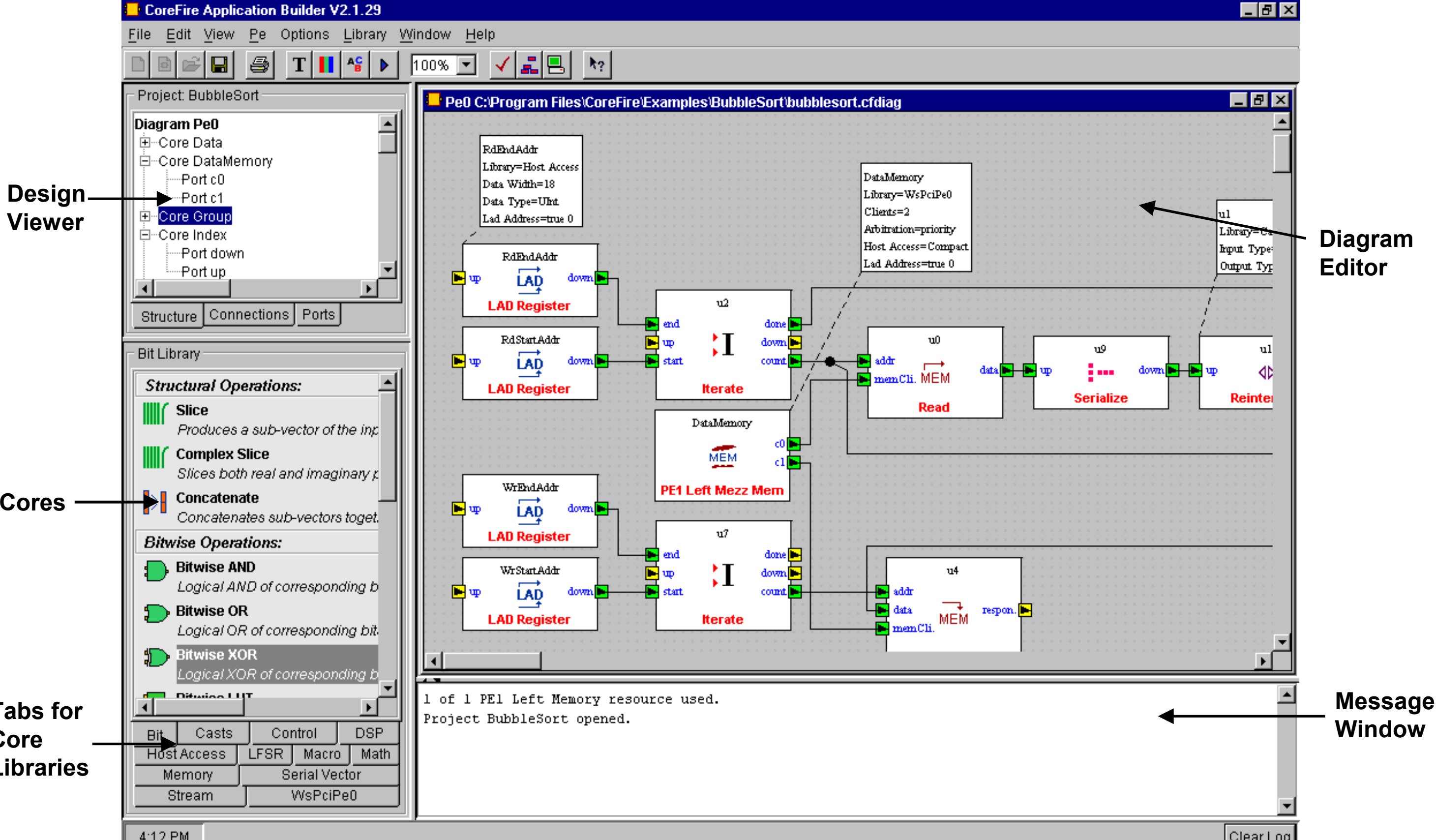


Figure 7-2: CoreFire™ Application Builder—Main Interface

Below are brief descriptions of the CoreFire™ libraries, including those specific to the ECL/PECL I/O card. ECL/PECL board libraries 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*. *Reinterpretin*g 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, Ws2VmePex Libraries

Cores in the Ws2VmePe libraries are dedicated to designs for PEs 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, Ws2PciPex Libraries

Cores in the Ws2PciPe libraries are dedicated to designs for PEs 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

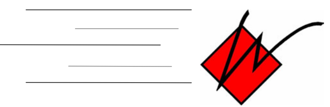
Cores in the FbPciPe0 Library are dedicated to designs for PE0 on a FIREBIRD™ / PCI motherboard. It contains cores for memory, channels between PEs, and channels between PEs and I/O cards.

WsEclPe, Ws2EclPe Libraries

Cores in the WsEclPe Library are specific to PE designs on an ECL/PECL 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. The Ws2EclPe Library contains cores for use with an ECL/PECL I/O card mounted on a WILDSTAR™-II motherboard.

FbEclPe Library

Cores in the FbEclPe Library are specific to PE designs on an ECL/PECL 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. ECL/PECL I/O CARD VHDL MODELS REFERENCE

8.1 WILDSTAR™ ECL/PECL I/O Card VHDL Model Overview

This chapter discusses the VHDL Model of the ECL/PECL I/O card as it fits into the simulation model of the WILDSTAR™ and WILDSTAR™-II host systems. Since this VHDL model is designed to support interfaces to both the ECL and PECL I/O boards, signal names and components that refer to the ECL I/O board are equally usable on the PECL I/O board.

Due to variations in motherboard architecture, several of the interfaces described in this chapter are board-specific: some are used only with WILDSTAR™ and FIREBIRD™ boards, while others are specific to WILDSTAR™-II. See section 8.2.2 for a detailed listing of interface compatibilities.

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 having a WILDSTAR™-II motherboard can only use the Mem_Mux components.

8.1.1 Block Diagrams

The following block diagrams illustrate the structure, layout, and connectivity of the WILDSTAR™/VME, WILDSTAR™-II /VME and PCI, FIREBIRD™/PCI, and WILDSTAR™ ECL/PECL I/O card architectures.

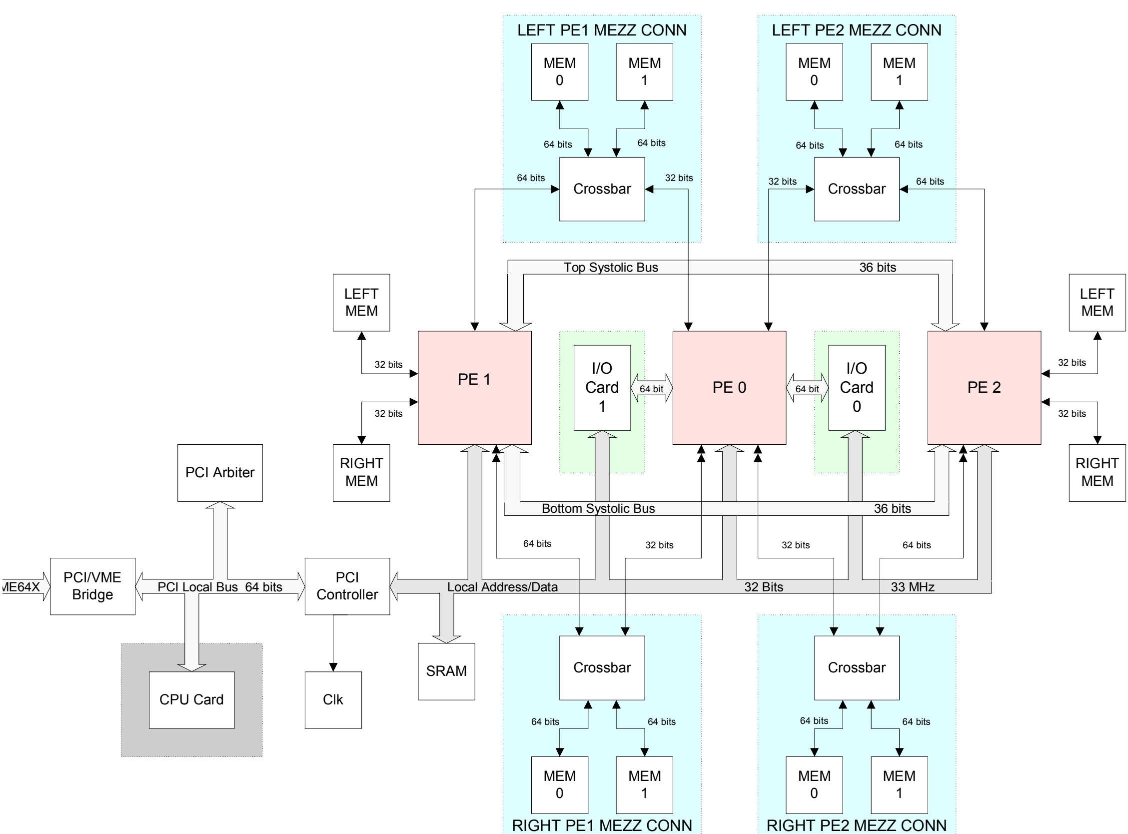


Figure 8-1: WILDSTAR™/VME Block Diagram

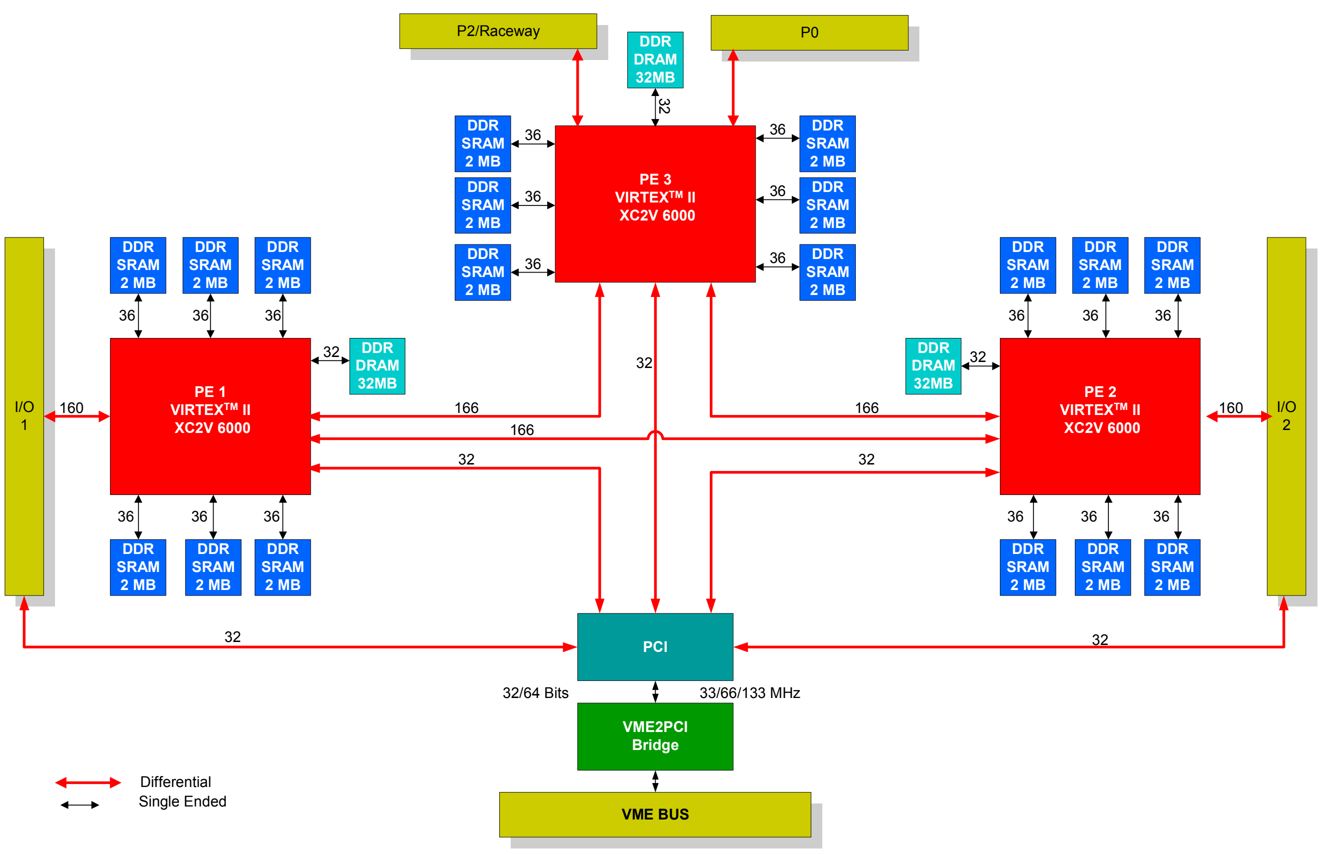
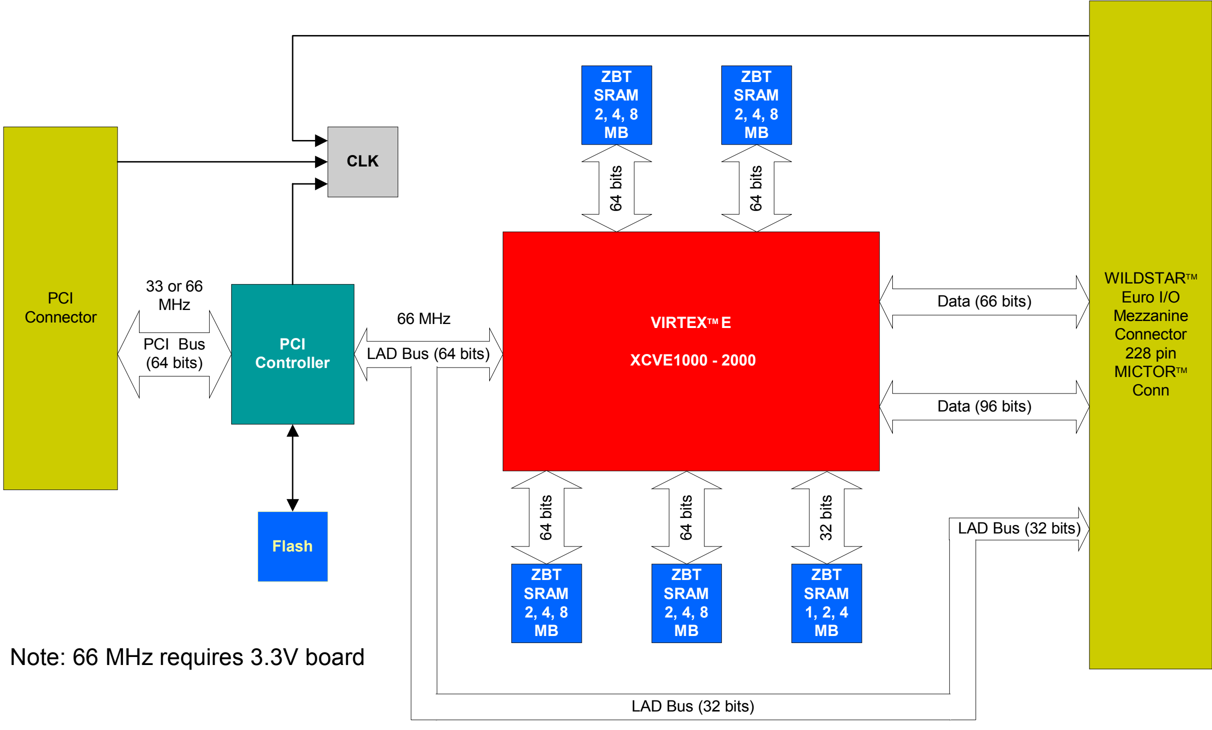


Figure 8-2: WILDSTAR™-II /VME Block Diagram



Copyright 2000-2001 Annapolis MIcro Systems, Inc.

Figure 8-3: FIREBIRD™/PCI Block Diagram

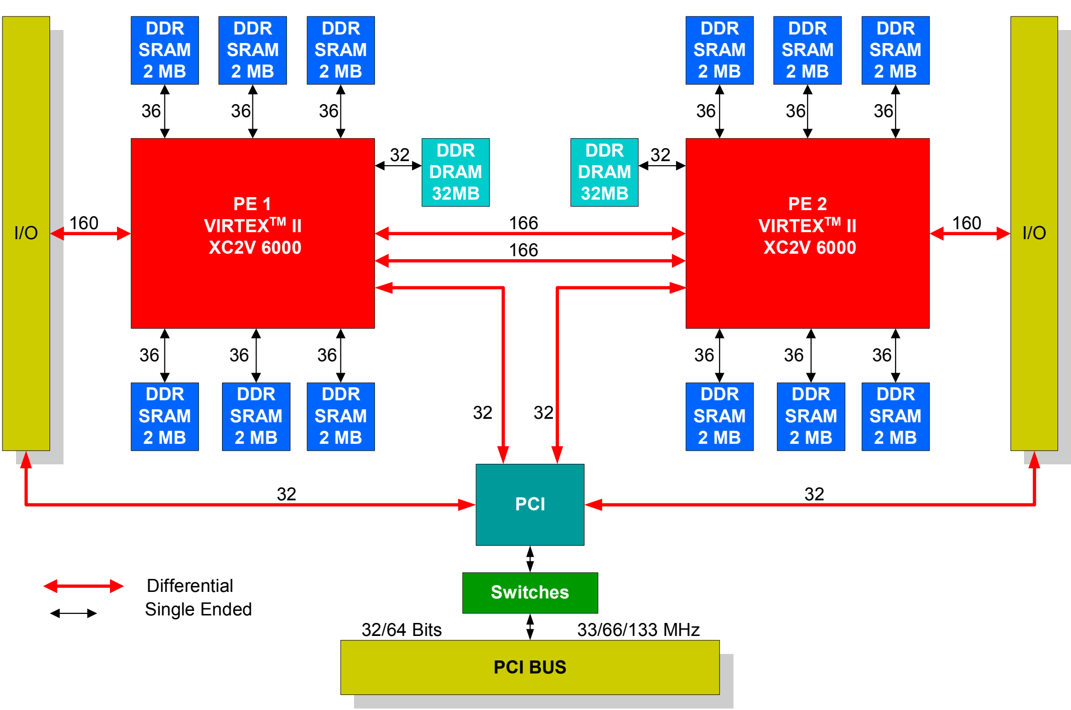


Figure 8-4: WILDSTAR™-II /PCI Block Diagram

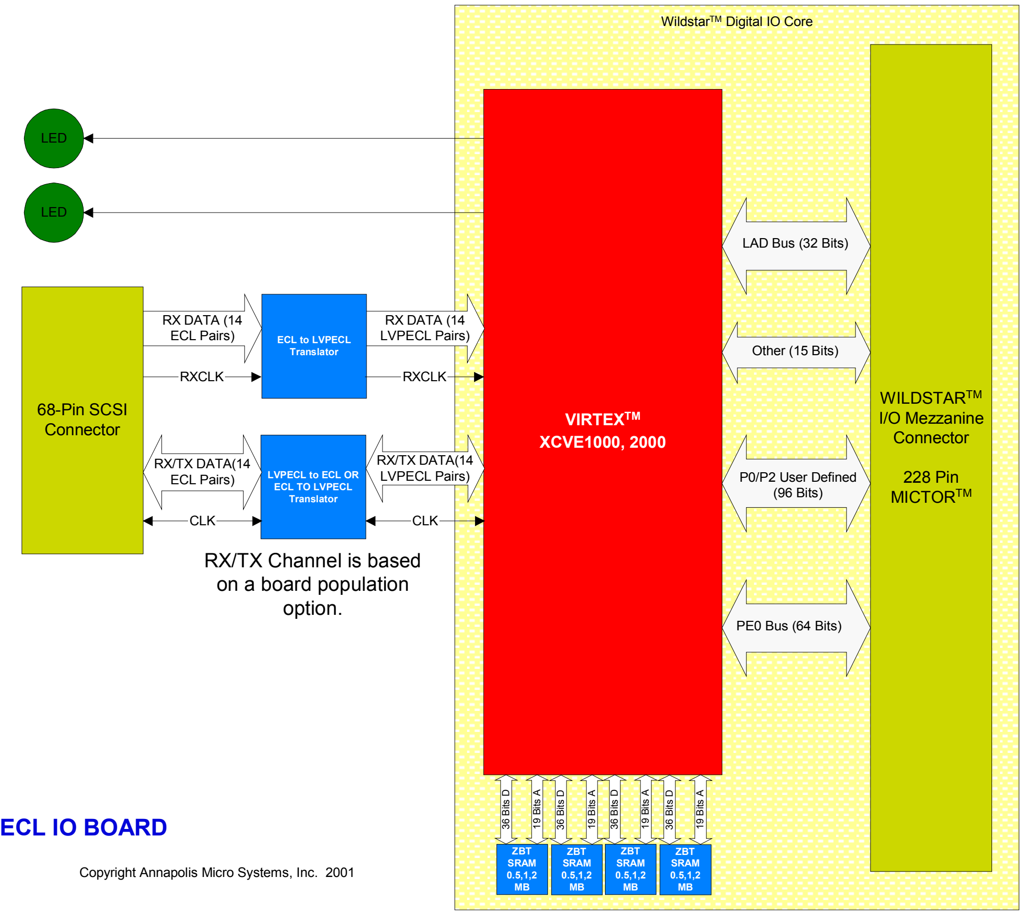


Figure 8-5: WILDSTAR™ ECL/PECL I/O Card Block Diagram

8.2 ECL/PECL I/O Card PE Model

8.2.1 PE Pad Definitions and Locations

The definitions and locations of the ECL/PECL I/O card PE pads are contained in the PE pad location data file (ecl_io_pe_pad_loc.dat), which is used by the Xilinx® Alliance tools to place the PE pads in the proper locations.

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 ECL/PECL I/O Card PE Interface Components

Due to motherboard architecture differences, several of the ECL/PECL I/O card interfaces will vary depending on the type of motherboard used with the 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
- ECL_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
- ECL_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.

8.2.3 ECL/PECL I/O Card PE Components

8.2.3.1 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 below. The "User_In" port signals should be used as the various clock signals in the design.

Table 8-1: Clock Standard Interface Component Port Signals

Signal Name	Width	Dir	Description
Pads.Clocks.M_Clk	1	Input	Memory clock pad signal; source can be selected from a user programmable clock, one of the ECL/PECL I/O card clocks.
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	User clock pad signal; source can be selected from a user programmable clock, or one of the ECL/PECL I/O card clocks.
Pads.Clocks.RACE_Clk	1	Input	Raceway Clock off the P2 backplane on VME systems
Pads.Clocks.U_Clk_M_Src	1	Output	Clock signal sent to the motherboard to become the global source for M_Clk across the board.
Pads.Clocks.U_Clk_U_Src	1	Output	Clock signal sent to the motherboard to become the global source for U_Clk across the board.
User_In.K_Clk	1	Output	User interface to the K_Clk pad signal. This signal will always be 33MHz when the WSPD™ I/O Card is attached to a WILDSTAR™/VME; however this signal can be 33MHz or 66MHz when attached to a FIREBIRD™/PCI.
User_In.K_Clk_2x	1	Output	User interface to the K_Clk pad signal; this clock is 66MHz. This signal should only be used on a WILDSTAR™/VME.
User_In.K_Clk_Locked	1	Output	K_Clk DLL lock flag; indicates DLL has achieved a lock when '1'
User_Out.K_Clk_DllRst	1	Input	User interface which allows manual reset of the K_Clk DLL.
User_In.M_Clk	1	Output	User interface to the M_Clk pad signal; source can be selected from a user programmable clock, or one of the I/O cards.
User_In.M_Clk_2x	1	Output	User interface to the M_Clk pad signal; source can be selected from a user programmable clock, or one of the I/O cards. This signal is the doubled output from the DLL.
User_In.M_Clk_90 User_In.M_Clk_180 User_In.M_Clk_270 User_In.M_Clk_DV	1 each	Output	Additional DLL outputs derived from the M_Clk pad signal. The DV is the Divided clock output which can be set by the M_CLK_DLL_DIVISOR generic. **These signals are not routed through a global clock buffer.**
User_In.M_Clk_Locked	1	Output	M_Clk DLL lock flag; indicates DLL has achieved a lock when '1'
User_Out.M_Clk_DllRst	1	Input	User interface which allows manual reset of the M_Clk DLL.
User_Out.Ext_M_Clk	1	Input	External M Clock Source
User_In.U_Clk	1	Output	User interface to the U_Clk pad signal; source can be selected from a user programmable clock, or one of the I/O cards
User_In.U_Clk_2x	1	Output	User interface to the U_Clk pad signal; source can be selected from a user programmable clock, or one of the I/O cards. This signal is the doubled output from the DLL.

Signal Name	Width	Dir	Description
User_In.U_Clk_90 User_In.U_Clk_180 User_In.U_Clk_270 User_In.U_Clk_DV	1 each	Output	Additional DLL outputs derived from the U_Clk pad signal. The DV is the Divided clock output which can be set by the U_CLK_DLL_DIVISOR generic. **These signals are not routed through a global clock buffer.**
User_In.U_Clk_Locked	1	Output	U_Clk DLL lock flag; indicates DLL has achieved a lock when '1'
User_Out.U_Clk_DllRst	1	Input	User interface which allows manual reset of the U_Clk DLL.
User_Out.Ext_U_Clk	1	Input	External U Clock Source
User_In.RACE_Clk	1	Output	User interface to the RACE_Clk pad signal; source can be selected from a user programmable clock, or one of the I/O cards
User_In.RACE_Clk_2x	1	Output	User interface to the RACE_Clk pad signal; source can be selected from a user programmable clock, or one of the I/O cards. This signal is the doubled output from the DLL.
User_In.RACE_Clk_90 User_In.RACE_Clk_180 User_In.RACE_Clk_270 User_In.RACE_Clk_DV	1 each	Output	Additional DLL outputs derived from the RACE_Clk pad signal. The DV is the Divided clock output which can be set by the RACE_CLK_DLL_DIVISOR generic. **These signals are not routed through a global clock buffer.**
User_In.RACE_Clk_Locked	1	Output	RACE_Clk DLL lock flag; indicates DLL has achieved a lock when '1'
User_In.Sky_Clk	1	Output	User interface to the Sky_Clk pad signal; source can be selected from a user programmable clock, or one of the I/O cards
User_In.Sky_Clk_2x	1	Output	User interface to the Sky_Clk pad signal; source can be selected from a user programmable clock, or one of the I/O cards. This signal is the doubled output from the DLL.
User_In.Sky_Clk_90 User_In.Sky_Clk_180 User_In.Sky_Clk_270 User_In.Sky_Clk_DV	1 each	Output	Additional DLL outputs derived from the Sky_Clk pad signal. The DV is the Divided clock output which can be set by the SKY_CLK_DLL_DIVISOR generic. **These signals are not routed through a global clock buffer.**
User_In.Sky_Clk_Locked	1	Output	Sky_Clk DLL lock flag; indicates DLL has achieved a lock when '1'

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

- **K_CLK_DLL_OUT**: When set to USE_1x, the User_In.K_Clk signal is driven by the 1x output of the K Clock DLL. User_In.K_Clk_2x is tied low. When set to USE_2x, the User_In.K_Clk_2x signal is driven by the 2x output of the K Clock DLL. User_In.K_Clk is tied low. When set to BOTH, both User_In.K_Clk and User_In.K_Clk_2x are driven by the 1x and 2x K clock DLL outputs respectively.

- **M_CLK_DLL_OUT**: When set to USE_1x, the User_In.M_Clk signal is driven by the 1x output of the M Clock DLL. User_In.M_Clk_2x is tied low. When set to USE_2x, the User_In.M_Clk_2x signal is driven by the 2x output of the M Clock DLL. User_In.M_Clk is tied low. When set to BOTH, both User_In.M_Clk and User_In.M_Clk_2x are driven by the 1x and 2x M clock DLL outputs respectively.

- **U_CLK_DLL_OUT**: When set to USE_1x, the User_In.U_Clk signal is driven by the 1x output of the U Clock DLL. User_In.U_Clk_2x is tied low. When set to USE_2x, the User_In.U_Clk_2x signal is driven by the 2x output of the U Clock DLL. User_In.U_Clk is tied low. When set to BOTH, both User_In.U_Clk and User_In.U_Clk_2x are driven by the 1x and 2x U Clock DLL outputs respectively.

- **RACE_CLK_DLL_OUT**: When set to USE_1x, the User_In.RACE_Clk signal is driven by the 1x output of the RACE Clock DLL. User_In.RACE_Clk_2x is tied low. When set to USE_2x, the User_In.RACE_Clk_2x signal is driven by the 2x output of the RACE Clock DLL. User_In.RACE_Clk is tied low. When set to BOTH, both User_In.RACE_Clk and User_In.RACE_Clk_2x are driven by the 1x and 2x RACE Clock DLL outputs respectively.

- **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.

- **M_CLK_SOURCE:** When set to OSCILLATOR, the User_In.M_Clk and the User_In.M_Clk_2x signals are driven by the motherboard oscillator. When set to RACE_CLK or EXT_CLOCK, User_In.M_signal is driven by RACEWay clock inputs or the signal on User_In.Ext_M_Clk. The 1x signal is also sent back to the mail board to become the source of M_Clk for the entire board.

- **U_CLK_SOURCE:** When set to OSCILLATOR, the User_In.U_Clk and the User_In.U_Clk_2x signals are driven by the motherboard oscillator. When set to RACE_CLK or EXT_CLOCK, User_In.M_signal is driven by RACEWay clock inputs or the signal on User_In.Ext_U_Clk. The 1x signal is also sent back to the mail board to become the source of U_Clk for the entire board.

- **M_CLK_USE_BUFG**: When set to TRUE, User_In.M_Clk and/or User_In.M_Clk_2x are routed using one of the four global clocking networks, and corresponding global clock buffer, on the FPGA. If M_CLK_DLL_OUT is set to USE_1x or USE_2x then only one global clock buffer is used. If M_CLK_DLL_OUT is set to BOTH then two global clock buffers are used.

- **RACE_CLK_USE_BUFG**: When set to TRUE, User_In.RACE_Clk and/or User_In.RACE_Clk_2x are routed using one of the four global clocking networks, and corresponding global clock buffer, on the FPGA. If RACE_CLK_DLL_OUT is set to USE_1x or USE_2x, then only one global clock buffer is used. If RACE_CLK_DLL_OUT is set to BOTH, then two global clock buffers are used.

- **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".

- **RACE_CLK_DLL_DIVISOR**: This string generic sets the divisor of the RACE_Clk DLL. The resultant divided output will be clocked out on the Clocks_In.RACE_Clk_DV signal. Valid RACE_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".

- **SKY_CLK_DLL_DIVISOR**: This string generic sets the divisor of the SKY_Clk DLL. The resultant divided output will be clocked out on the Clocks_In.SKY_Clk_DV signal. Valid SKY_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.

8.2.3.2 LAD Interfaces

The Local Address/Data Bus is the primary means by which the ECL/PECL I/O PE communicates with the host system. There are two different LAD bus specifications, and therefore two different LAD bus interfaces. ECL/PECL I/O boards which have a WILDSTAR™ /VME or FIREBIRD™ /PCI motherboard should use the LAD_Mux_IF described in section 8.2.3.2.1 below. ECL/PECL I/O boards which have a WILDSTAR™-II motherboard should use the WSII_LAD_Bus_Std_IF described in section 8.2.3.2.2 below.

8.2.3.2.1 LAD Mux Interface (LAD_Mux_IF)

The LAD Mux interface (LAD_Mux_IF) uses the LAD bus path between the PCI Controller and ECL/PECL 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.

8.2.3.2.2 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 ECL/PECL 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

Signal Name	Width	Dir	Description
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.

8.2.3.2.2.1 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.

8.2.3.2.2.2 Register Space Transactions

A typical LAD bus register write access is illustrated in the timing diagram labeled Figure 8-6. 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.

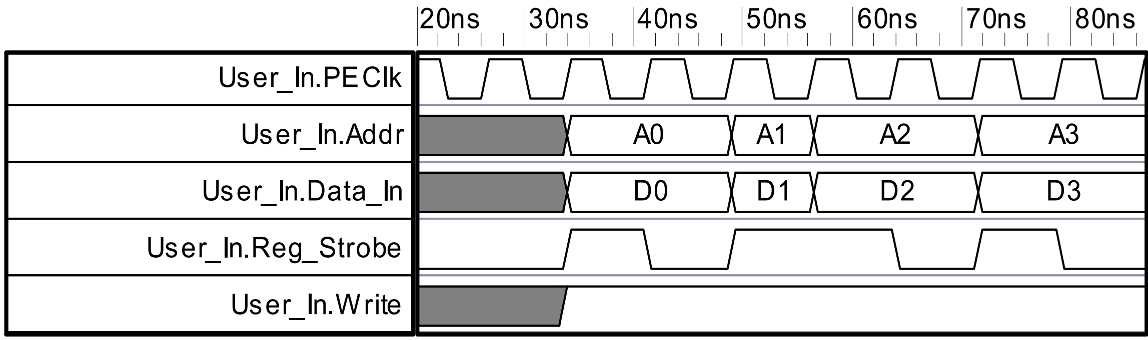


Figure 8-6: Typical LAD Bus Write Cycle from Register Space

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.

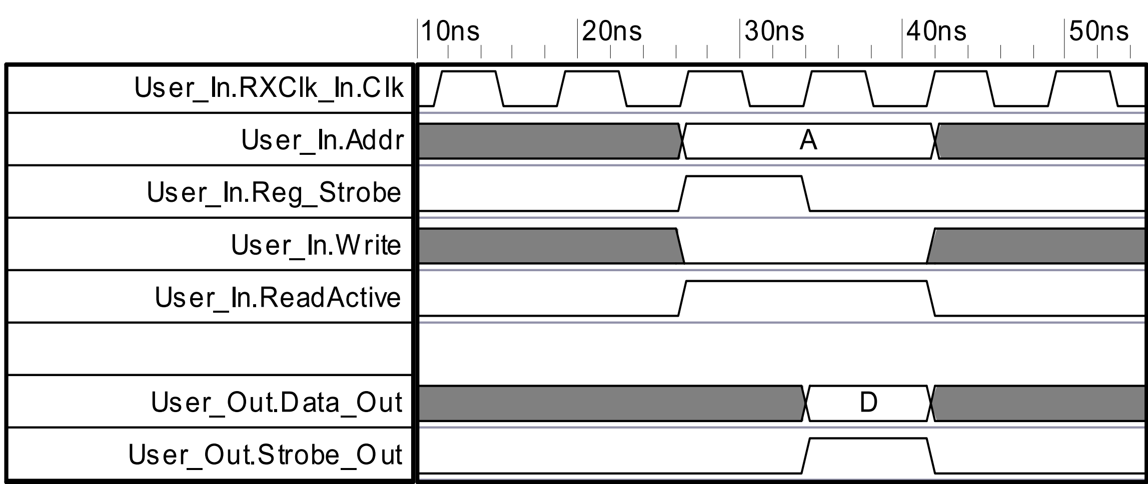


Figure 8-7: Shortest Possible Register Space LAD Bus Read Cycle

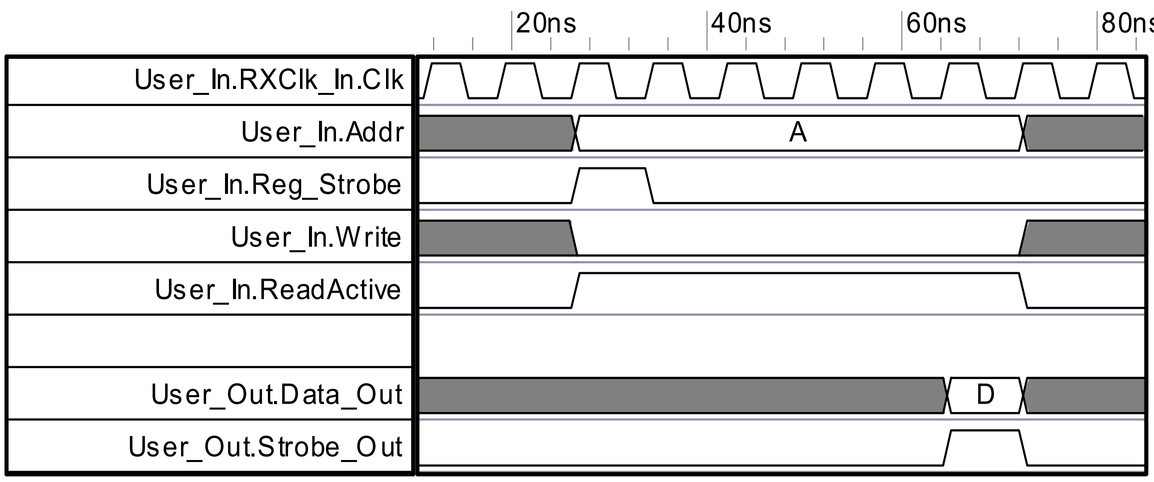


Figure 8-8: Longer Register Space LAD Bus Read Cycle

8.2.3.3 LED Standard Interface (LED_Std_IF)

The LED_Std_IF standard interface provides the user interface to the light emitting diodes (LEDs) of the ECL/PECL I/O card PE device. The LEDs 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 the table below. The active low "User_Out" signals should be used to drive the LED of the PE device.

Table 8-5: LED Standard Interface Component Port Signals

Signal Name	Width	Dir	Description
Pads	2	Output	LED pad signals
User_Out_n	2	Input	User interface to the LED

8.2.3.4 ECL/PECL Transmitter Standard Interface (ECL_Transmitter_STD_IF)

The ECL/PECL Transmitter Standard interface provides the user interface to the 14 data lines and one clock line of a transmitting ECL/PECL channel. The port signals for the ECL/PECL transmitter are described in the table below.

Table 8-6: ECL/PECL Transmitter Interface Component Port Signals

Signal Name	Width	Dir	Description
Pads.Clock	2	Output	Differential LVPECL transmit Clock signal.
Pads.Data	28	Output	14 pairs of Differential LVPECL data signals.
Global_Reset	1	Input	Reset (or set) signal; usually connected to the GSR input of a STARUP_VIRTEX component
Clock	1	Input	Transmit Clock; Also used to clock out all data
User_Out.Data_Out	14	Input	Data to be sent out the ECL transmitter

8.2.3.5 ECL/PECL Receiver Standard Interface (ECL_Receiver_STD_IF)

The ECL/PECL Receiver Standard interface provides the user interface to the 14 data lines and 1 clock line of a receiving ECL/PECL channel. To clock in the receive data the user can choose to recover the ECL/PECL connector clock input using a DLL, or to supply a clock to the interface. The port signals for the ECL/PECL transmitter are described in the tables below.

Table 8-7: ECL/PECL Receiver Interface Component Port Signals when CLOCK_SOURCE=ECL_CONNECTOR and CLK_DLL_TYPE=HIGH_FREQ or LOW_FREQ

Signal Name	Width	Dir	Description
Pads.Clock	2	Input	Differential LVPECL receive Clock signal.
Pads.Data	28	Input	14 pairs of Differential LVPECL data signals.
Global_Reset	1	Input	Reset (or set) signal; usually connected to the GSR input of a STARUP_VIRTEX component
Clock_In.Clk	1	Output	Recovered Receive Clock 1x DLL output
Clock_In.Clk_2x	1	Output	Recovered Receive Clock 2x DLL output
Clock_In.Clk_90	1	Output	Recovered Receive Clock 90° DLL output
Clock_In.Clk_180	1	Output	Recovered Receive Clock 180° DLL output
Clock_In.Clk_270	1	Output	Recovered Receive Clock 270° DLL output
Clock_In.Clk_DV	1	Output	Recovered Receive Clock Divided DLL output
Clock_In.Locked	1	Output	Recovered Receive Clock DLL locked output
Clock_Out.Reset	1	Input	Recovered Receive Clock DLL reset input
Clock_Out.Ext_Clk	1	Input	Unused
User_In.Data_In	14	Output	Data received from the ECL channel

Table 8-8: ECL/PECL Receiver Interface Component Port Signals when CLOCK_SOURCE=ECL_CONNECTOR and CLK_DLL_TYPE=NONE

Signal Name	Width	Dir	Description
Pads.Clock	2	Input	Differential LVPECL receive Clock signal.
Pads.Data	28	Input	14 pairs of Differential LVPECL data signals.
Global_Reset	1	Input	Reset (or set) signal; usually connected to the GSR input of a STARUP_VIRTEX component
Clock_In.Clk	1	Output	Recovered Receive Clock pad output
Clock_In.Clk_2x	1	Output	Unused
Clock_In.Clk_90	1	Output	Unused
Clock_In.Clk_180	1	Output	Unused
Clock_In.Clk_270	1	Output	Unused
Clock_In.Clk_DV	1	Output	Unused
Clock_In.Locked	1	Output	Unused
Clock_Out.Reset	1	Input	Unused
Clock_Out.Ext_Clk	1	Input	Unused
User_In.Data_In	14	Output	Data received from the ECL channel

Table 8-9: ECL/PECL Receiver Interface Component Port Signals when CLOCK_SOURCE=EXT_CLOCK

Signal Name	Width	Dir	Description
Pads.Clock	2	Input	Differential LVPECL receive Clock signal.

Signal Name	Width	Dir	Description
Pads.Data	28	Input	14 pairs of Differential LVPECL data signals.
Global_Reset	1	Input	Reset (or set) signal; usually connected to the GSR input of a STARUP_VIRTEX component
Clock_In.Clk	1	Output	Driven directly by User_In.Ext_Clk
Clock_In.Clk_2x	1	Output	Unused
Clock_In.Clk_90	1	Output	Unused
Clock_In.Clk_180	1	Output	Unused
Clock_In.Clk_270	1	Output	Unused
Clock_In.Clk_DV	1	Output	Unused
Clock_In.Locked	1	Output	Unused
Clock_Out.Reset	1	Input	Unused
Clock_Out.Ext_Clk	1	Input	External User clock input. Used to clock receive data into the PE.
User_In.Data_In	14	Output	Data received from the ECL channel

This interface also contains several generic to control the clocking and DLL setup. These generics are described below.

- **CLOCK_SOURCE:** This generic can be set to ECL_CONNECTOR or EXT_CLOCK. When set to ECL_CONNECTOR the interface uses the recovered receive clock for clocking in data, and drives this clock on the Clock_Out port. When set to EXT_CLOCK the interface uses clock on the Clock_In port for clocking in data, and drives this clock on the Clock_Out port.
- **USE_BUFG:** This generic can be set to TRUE or FALSE. When set to TRUE, the output of the receive clock DLL is sent through a global clock buffer. To reduce clock skew and place and route times this generic should be set to TRUE. USE_BUFG is ignored if CLOCK_SOURCE is set to EXT_CLOCK.
- **CLK_DLL_TYPE:** This generic can be set to HIGH_FREQ, LOW_FREQ or NONE. When set to HIGH_FREQ or LOW_FREQ, the recovered clock is first set through a high or low frequency DLL, respectively. When set to NONE, a DLL is not used so Clock_In is driven directly by the receive clock pad. CLK_DLL_TYPE is ignored if CLOCK_SOURCE is set to EXT_CLOCK.
  DLL_DIVIDE: This string generic sets the divisor of the receive clock DLL. The resultant divided output will be driven out on the Clock_Out.Clk_DV signal. Valid DLL_DIVIDE values 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”.

8.2.3.6 Mem36_Mux_IF (Priority and Fair)

There are four independent Mem36_Mux_IF memory ports on the ECL/PECL I/O card. The memory port signals are described in Table 8-10. The Mem32 Mux signals are described in Table 8-11.

Table 8-10: Mem 36 Mux Interface Component Port Signals

Signal Name	Width	Dir	Description
Mclk	1	Bi	Memory bus clock signal
Reset	1	Input	Reset (or set) signal; usually connected to the GSR input of a STARUP_VIRTEX component
Mem_Pads	Record	Bi	Please reference Table 7-8.
Clients	Record	Bi	Array of Mem36_Mux_vector records which represent every design unit that accesses memory attached to this mux. Please see Table 7-8, which describes the Clients record.

Table 8-11: Mem 36 Mux Vector Signals

Signal Name	Width	Dir	Description
Clients(I).Addr	32	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

8.2.3.7 Motherboard I/O Connectors

The ECL/PECL I/O device has a number of data bits connecting the ECL/PECL 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 ECL/PECL I/O PE to the VME backplane. If the ECL/PECL 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 ECL/PECL 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.

8.2.3.7.1 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 8-12. 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: 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 8-13: 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

8.2.3.7.2 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 8-14: 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 8-15: PE0 Connector Standard Interface Motherboard Signal Mapping

Motherboard	Motherboard Interface	Motherboar d 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

8.2.3.7.3 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 8-16: 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 8-17: 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

8.3 ECL/PECL Cables

Each ECL/PECL I/O card has two 14-bit differential ECL ports that can be connected to an ECL port on the same card, on a second ECL/PECL I/O card on the same board, or between ECL/PECL I/O cards on a second board (cable not included with card). In order to simulate an ECL/PECL cable, the ECL/PECL I/O VHDL model includes a cable package.

The ecl_glink_cables_pkg.vhd, located in the "annapolis/shared/system/connectors /ecl" directory, contains a list of predefined signals to use to connect ECL ports on various cards.

8.3.1 ECL/PECL Cables

The signals of the ECL/PECL I/O cable are described in Table 8-18.

Table 8-18: ECL/PECL I/O Cable Signals

Signal Name	Width	Dir	Description
Clock	1	Bi-Dir	ECL transmit or receive clock
Data	17	Bi-Dir	ECL transmit or receive data

Ten predefined cables are instantiated in the ecl_cables_pkg.vhd file. They are named ECL_Cable_x, where *x* is 0 through 9.

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.

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 8-9 illustrates the Mux Library Scheme.

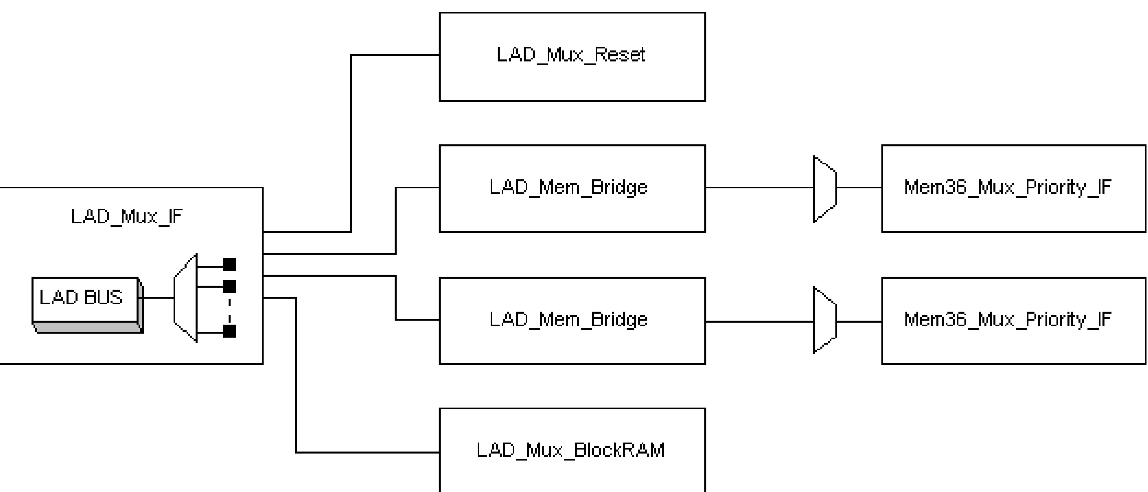


Figure 8-9: General Mux Library Scheme

8.4.1 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 8-19:

Table 8-19: 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.

i

INFORMATION NOTE

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

8.4.1.1 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

i

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

8.4.1.2 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.

8.4.1.3 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*.

8.4.1.4 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.

8.4.1.5 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.

8.4.1.6 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.

8.4.1.7 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.

8.4.1.8 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 8-20 shows each LAD Mux component, the number of address spaces each component consumes, and the mask created for each component.

Table 8-20: 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 8-21
LAD_Mux_CRegFile	2^{L2Num}	See Table 8-21
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 8-21: 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.

8.4.1.8.1 LAD_Mux_Reset

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

8.4.1.8.2 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 8-22: 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.

8.4.1.8.3 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 8-23: 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**

Incorrect Base Choice:

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 address 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.)

Correct Base Choice
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 8-23 and the examples above to calculate correct base selections for the LAD_Mux_BlockRAM64.)

8.4.2 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®.

8.4.2.1 Memory Arbitration Schemes

There are two types of memory arbitration schemes that can be used. Both types are described below.

8.4.2.1.1 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 8-24 for a complete description of these generics.

Table 8-24: 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 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 of who will be acknowledged first 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.

Generic Name	Values	Description
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.

8.4.2.1.2 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 8-25 for a complete description of these generics.

Table 8-25: 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.

8.4.2.2 Mem36_Mux Control

The Mem36_Mux record definitions are shown in Table 8-26:

Table 8-26: Mem36_Mux Record Signals

Signal Name	Width	Dir	Description
Addr	32	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 32 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 are indicative of 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.

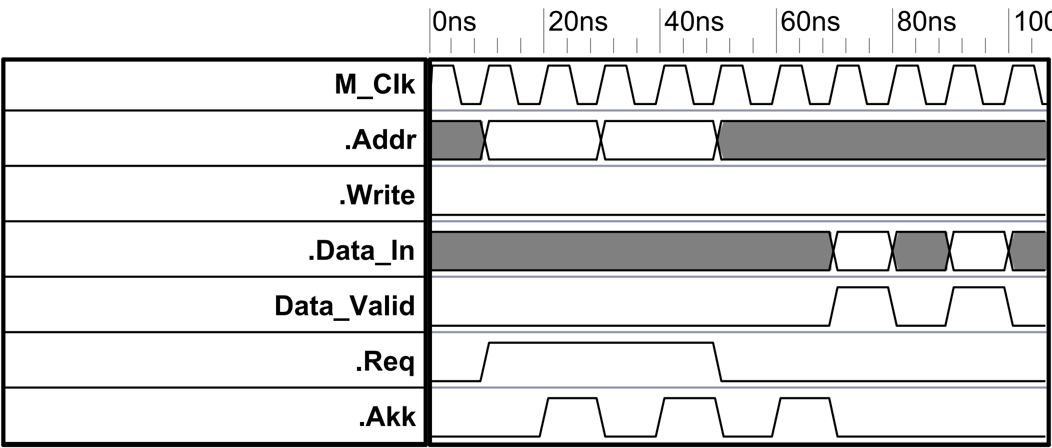


Figure 8-10: Mem36_MuxRead Cycle

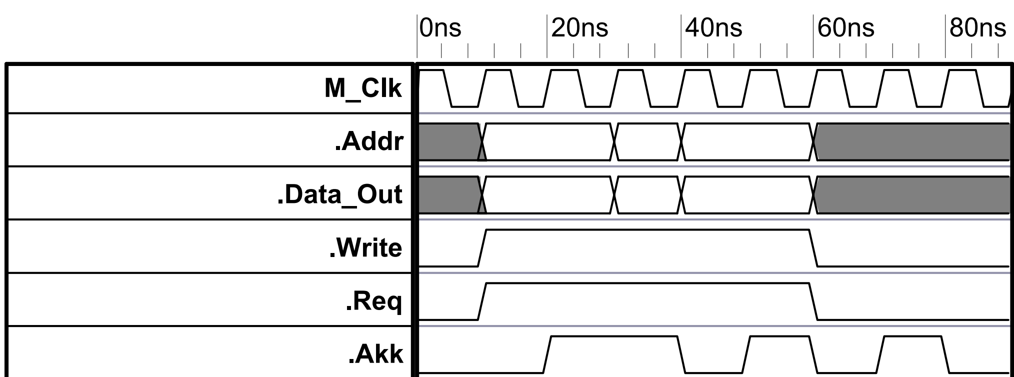


Figure 8-11: Mem36_MuxWrite Cycle

8.4.3 Programmed I/O Memory Bridge

8.4.3.1 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 the 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).

8.4.3.2 Assigning the BASE to the LAD Memory Bridge

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, a “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.

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 8-27: Address in the LAD_Mem36_Bridge

Addresses	Consumed By
0x000-0x0FF	LAD_Mux_BlockRAM
0x100-0x103 0x102-0x1FF	LAD_Mux_CRegfile w/L2Num = 2 Reserved

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

Table 8-28: Example Base Address Table for the Mem36 Bridge

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

<u>Incorrect Base Choice:</u>
Because this component contains a LAD_Mux_BlockRAM, the same problems described in Section 8.4.1.8.3 will occur. Therefore, no information from the LAD bus will be written or read to or from the memories.

<u>Correct Base Choice</u>
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 memory 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.

8.4.4 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 8-12 illustrates the General DMA Mux Library Scheme.

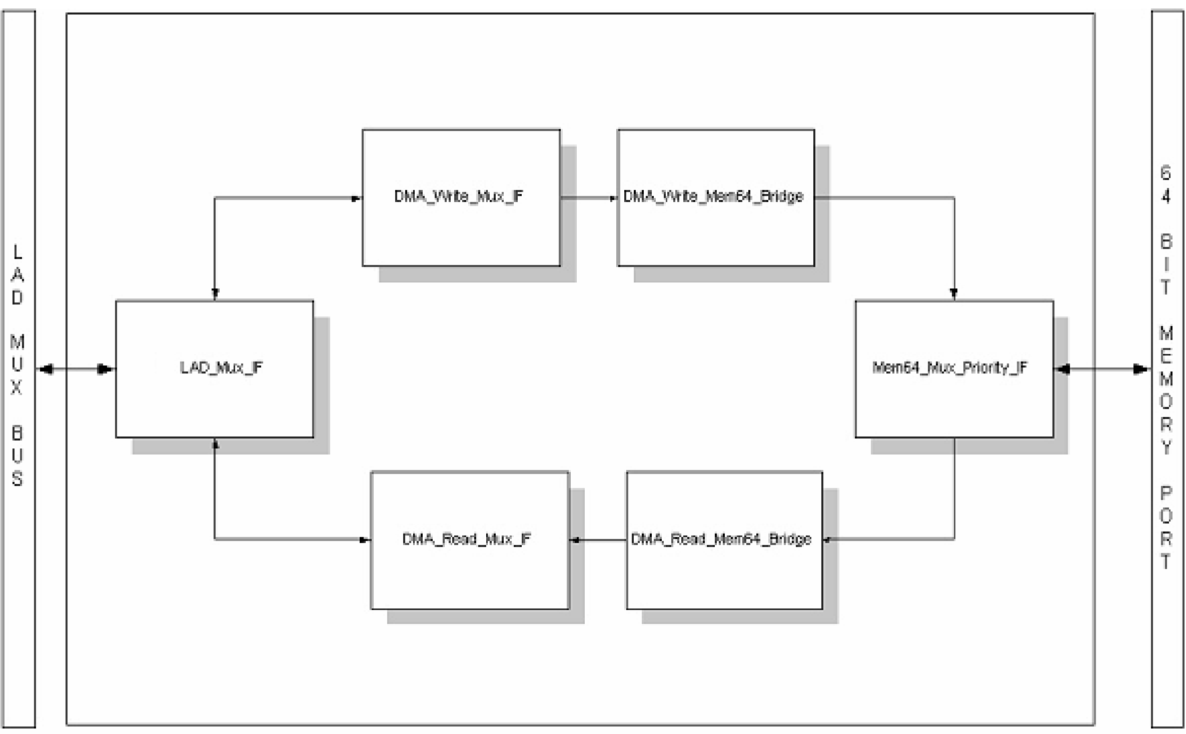


Figure 8-12: General DMA Mux Library Scheme

8.4.4.1 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 Table 8-29:

Table 8-29: 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 32 bits.
Fifo_Status	5	Output	User interface which indicate 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 reach the ¾ full mark. The ¾ mark is used to prevent over running the input FIFOs

The LAD port is a unique LAD_Mux_Vector, which is described in Table 8-19: 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.

The DMA_Write_Mux_IF has one generic which is described below

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

8.4.4.2 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 8-30.

Table 8-30: LAD_DMA_Read_Mux_Vector Record Signals

Signal Name	Width	Dir	Description
Data_Out	32	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 8-19: 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 which is described below

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

8.4.5 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 8-19: 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, as well as the RELOAD_EN generic, are described in sections 8.4.5.1 and 8.4.5.2.

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 8-31.

Table 8-31: 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.

While the memory bridges all operate similarly, they are described in separate sections to avoid confusion.

8.4.5.1 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 in Section 8.4.5, 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 exert 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;
```

8.4.5.1.1 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.

8.4.5.2 DMA Read Bridge

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 in Section 8.4.5, 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 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;
```

8.4.5.2.1 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.

8.4.5.3 Assigning the BASE to the LAD Memory Bridges

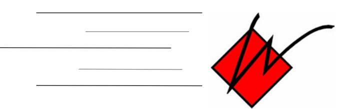
The DMA Bridges each contain a LAD64_Mux_CRegFile with the L2Num set to two. This means that there are a total of four address spaces consumed by these components.

8.4.6 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.

THIS PAGE INTENTIONALLY LEFT BLANK



9. ECL/PECL I/O CARD VHDL GUIDE

The VHDL design cycle consists of several simple steps that help an application developer create, simulate, synthesize, and analyze an ECL/PECL I/O system design. This chapter describes how to create a design using the ECL/PECL I/O system VHDL models.

9.1 VHDL Design Cycle

i

INFORMATION NOTE
The ECL/PECL I/O VHDL model is not a standalone VHDL mode. In order to use the ECL/PECL I/O VHDL model a motherboard, such as WILDSTAR™/VME or FIREBIRD™/PCI, must be compiled in conjunction with the model.

9.1.1 ECL/PECL I/O Template VHDL Design Files

The VHDL template files are all found in the sub-directory "$IO_CARD_BASE/ ecl_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

There are three macro scripts which are necessary to compile the ECL/PECL I/O VHDL models. Two of the files, the **project_vcom.do** and **project_vsim.do**, are contained in the WILDSTAR™/VME and FIREBIRD™/PCI subdirectories. (Please refer to the WILDSTAR™/VME or FIREBIRD™/PCI hardware manual for more information.)

The third script, the **project_ecl_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 ECL/PECL simulation model.

9.1.1.2 PE VHDL Template files

9.1.1.2.1 ECL/PECL PE Architecture Template

On of the two template files, **ecl_template_arch.vhd** or **ecl_wsii_template_arch.vhd** should be used when creating an ECL/PECL I/O card design, since the template files have been pre-designed to contain most of the boilerplate interface logic 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. Ecl_template_arch.vhd should be use for designs with a WILDSTAR™ /VME or FIREBIRD™ motherboard and Ecl_wsii_tempate_arch.vhd for designs with a WILDSTAR™-II motherboard.

9.1.1.2.2 ECL/PECL I/O Connector Interface Template

The **ecl_if_template_arch.vhd** file is used simulate external connections to the ECL/PECL I/O card.

9.1.1.2.3 ECL/PECL I/O Backplane Interface Template

The **ecl_backplane_if_template_arch.vhd** file is used to simulate backplane connections to the ECL/PECL I/O card.

9.1.1.3 ECL/PECL I/O Configuration Templates

The ECL/PECL I/O card configuration files (**ecl_io_card_0_cfg.vhd** and **ecl_io_card_1_cfg.vhd**) must be modified to meet the needs of the design. The files 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_Mem_0**: (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_Mem_1**: (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_Mem_2**: (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_Mem_3**: (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_Ecl_IF**: The ECL/PECL interface architecture can be changed to the name used by a particular application.
- **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. Using the file browser, 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/ecl_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	ECL/PECL 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. Using the file browser, 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 ECL/PECL I/O 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 that 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 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	PE 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 ECL/PECL I/O 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 that must be modified in order for the makefile to produce the correct binary file for the ECL/PECL I/O card.

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 ECL/PECL I/O 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.

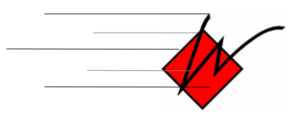
<design>.par	Area and timing results generated by the **par** tool
<design>.twr	Timing report generated by the **trce** tool

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 a .x86 file; on big-Endian machines (Sun, ALPHA™, VxWorks®), the binary file needs to be a .m68 file. This file is produced by the peutil program.

To run the peutil program under unix, execute
"<ANNAPOLIS_BASE>**\host\tools\peutil** <design>**.mcs** <design>**.m68"**
for each design in the project.



INDEX

A

API, 1-3
Application Builder Online Help, 7-1
Application Programming Interface, 1-3
architecture, 2-4

B

back plate, 3-2, *4-7*, *4-8*, *4-9*, *4-10*, *4-11*
Back Plate, 3-5

C

card 0, 2-1
clock, 2-1, 6-2, 6-3, 6-6
Clocks, 6-2
Compatibility, 4-1
connector plug, 4-3, *4-6*, *4-7*, *4-8*, *4-9*, *4-11*
connectors, 2-4, 4-3

E

ECL I/O, i, 1-1, 1-2, 1-3, *2-1*, 2-4, 2-5, 3-1, 3-2, 3-5, 3-6, 4-1, 4-2, 4-3, *4-5*, *4-6*, *4-7*, *4-8*, *4-9*, *4-10*, *4-11*, *4-12*, *6-1*, 6-2, 6-3, 6-6, 8-1, 8-6, 8-8, 8-17, 8-23, *9-1*, *9-2*, *9-4*, *9-5*
  Features, 2-1
  hardware, 3-1
  interface, 2-5
  Introduction, 2-1
  LEDs, 3-6
  pins, 2-4
ejectors, 4-3, *4-5*
Examples. See VHDL, See VHDL
External I/O, 1-3

F

FIREBIRD, i, 1-1, 1-2, 2-1, 2-3, 3-2, 3-5, 4-1, 4-2, *4-8*, *4-9*, *4-10*, *4-11*, 6-1
FPGA, 2-1, 7-1
Frequency Parameters, 6-6

G

General Specifications, 6-1

H

Hardware, 6-1
host software, 6-2

I

I/O Card 0, 1-3
I/O Card 1, 1-3
Icons, 1-2
Input/Output, 1-3
Introduction, 7-1

K

KCLK, 6-2, 6-3, 6-6

L

LED, 3-6, 4-3, *4-5*
LED Definition, 3-6
Lower-Level Cores, 7-1

M

Main Board, 1-3
MCLK, 6-2, 6-3, 6-6, 6-7
memory card, 3-1, 4-3, 4-4, *4-6*
mezzanine spacers, 4-3
ModelSim, 1-4
Mounting Block, 3-2

P

PE. See Processing Element
PE3, 2-1, 2-4, 6-2, 6-3, 6-6, 6-7
Physical Dimensions, 6-1
PMC, *8-43*, *9-5*
power, 4-2, *4-5*, *4-7*, *4-8*, *4-9*, *4-10*
Processing Element, *1-3*

R

Race++, 2-5

RACEway™, 2-5, 6-3
Register Space Transactions, 8-15

S

SRAM, 2-1
Standoff, 3-2
System requirements, 4-1

T

Temperature, 6-1
throughput, 2-5
Tools, 7-1
trace points, 7-1

U

UCLK, 6-2, 6-3, 6-6, 6-7
UCLK, 6-2

V

VHDL, iii, 1-2
VHDL Model, 8-1

W

WILDSTAR™ Host Software, *1-4*
WILDSTAR™ II, 1-1, 8-14
WILDSTAR™ Reference Manual, *4-4*, *4-6*, 6-2
WILDSTAR™/VME Block Diagram, 8-2
WILDSTAR™-II, 1-1
WSDP™/VME Front Panel, 3-4

X

XCLK, 6-3