

# L2 $\beta$ Design Report

A replacement and upgrade path for DØ Level 2 processor cards

Bob Hirosky

University of Virginia

Drew Baden

University of Maryland

Philippe Cros, Bernard Lavigne, Pierre Petroff

Laboratoire de l'Accélérateur Linéaire

version 2.0.1

November 7, 2001

The most recent version of this note may be found at the L2 $\beta$  website:

<http://galileo.phys.virginia.edu/~rjh2j/l2beta>

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Motivation</b>	<b>2</b>
<b>3</b>	<b>The L2<math>\beta</math> Strategy</b>	<b>4</b>
<b>4</b>	<b>Hardware Overview</b>	<b>6</b>
4.1	L2 $\beta$ SBC . . . . .	6
4.2	L2 $\beta$ 9U Adapter . . . . .	7
<b>5</b>	<b>System Requirements</b>	<b>7</b>
5.1	Processing Power . . . . .	7
5.2	Linux/RT Unix Operating System . . . . .	8
5.3	VME, TSI, MBus Devices . . . . .	8
5.3.1	Hardware Requirements . . . . .	8
5.3.2	Functional Requirements . . . . .	9
5.4	VME Interface . . . . .	10
5.4.1	Registers . . . . .	11
5.4.2	Description of Operation . . . . .	11
5.5	Mechanical and Electrical . . . . .	12
<b>6</b>	<b>Software</b>	<b>12</b>
6.1	Operating system, compiler, build environment . . . . .	13
6.1.1	Kernel Modifications . . . . .	13

6.2	Device Driver Software . . . . .	13
<b>7</b>	<b>Firmware</b>	<b>13</b>
7.1	PLX 9656 general configuration . . . . .	14
7.2	Local Bus Firmware . . . . .	15
7.3	MBus I/O and Arbitration Firmware . . . . .	16
7.4	DMA Firmware Functions . . . . .	18
7.4.1	Description of DMA operation . . . . .	18
7.5	PIO Firmware Functions . . . . .	19
7.5.1	Configuration . . . . .	19
7.5.2	PCI Transactions . . . . .	20
7.5.3	Magic Bus Transactions . . . . .	22
7.5.4	PIO and PCI Compliance . . . . .	23
7.5.5	DMA vs. PIO Priorities . . . . .	23
7.6	TSI Firmware Functions . . . . .	24
7.6.1	Description of Operation . . . . .	24
7.6.2	DØ Trigger Signals . . . . .	24
7.6.3	Communication with Global Level 2 . . . . .	24
7.6.4	Monitoring of Magic Bus Status Lines . . . . .	24
7.6.5	TSI Memory Registers . . . . .	25
7.6.6	Summary of new i/o functionality . . . . .	26
7.7	VME and Scaler Firmware . . . . .	29
7.8	Spy and Configuration Channels . . . . .	29
<b>8</b>	<b>Future Upgrade Paths</b>	<b>29</b>
<b>9</b>	<b>Acknowledgements</b>	<b>29</b>
<b>A</b>	<b>Magic Bus Connections and Arbitration</b>	<b>29</b>
<b>B</b>	<b>Document History</b>	<b>29</b>

## 1 Introduction

This document details a more contemporary model for the construction of the Level 2 processor cards. The new cards will be fully compatible with the present system components, including the L2Alpha processors. Additionally, this design provides a path to upgrade Level 2 with increased processing power and greater data throughput. The new design offers a minimum of engineering effort, and, through use of standard industrial processor cards and modern FPGAs, will provide a more flexible, upgradable, and more easily commissioned system.

## 2 Motivation

There are a number reasons motivating this new design of the Level 2 processor cards:

1. to quickly provide trigger processors to supplement shortfalls in available alpha boards;
2. to develop a source of processors for long term maintenance of the trigger;

3. to provide processing performance improvements during Run II for nodes that may suffer in performance at peak luminosities or for subsystems demanding more complex trigger processing;
4. and to develop a platform compatible with future enhancements for the higher luminosity running of Run IIb.

Any modernization or upgrade path using a monolithic processor card model would require an engineering effort equally large as the original L2Alpha efforts, if a more modern, but similar approach were pursued. Any such implementation would obviously suffer the fate of rapid obsolescence<sup>1</sup> as all modern computing devices. Furthermore, the prototyping and verification of the design would prove difficult (especially when considering faster clock speeds in more modern CPUs), expensive (in terms of engineering and prototyping), and taxing of manpower resources.

While the 500 MHz Alpha CPUs were superior in performance at the genesis of the project they no longer dominate the alternatives and are now surpassed by the latest processors. Our current processor boards as implemented cannot benefit from this march of technology.

At highest luminosities, the current system may be limited by CPU performance as more channels cross thresholds for clustering or fitting. Anticipating little room to improve the CPU efficiency of our trigger algorithms, there are several possibilities for dealing with data requiring with greater CPU performance than the L2Alphas can provide:

- Increase seed thresholds. For example, in the Calorimeter Electron and Jet Preprocessors [4], seed thresholds are fixed in hardware at Level 1, but higher software thresholds could be imposed in the processor code. The effect of this would be to reduce efficiency, especially for lower  $P_T$  objects.
- Increase the Level 1 trigger threshold or prescale. Again efficiencies are significantly lowered.
- Remove functionality from the algorithm. This further taxes the Level 3 trigger by decreasing background rejection at Level 2. The resulting functionality loss may combine with input bandwidth limitations to Level 3 to force raising thresholds at Level 1 or Level 2, also lowering efficiency.
- Add more Worker CPUs. It is possible to add additional workers to the Level 2 crates. In the case of certain detector preprocessors, the algorithm may be parallelized in a fairly straightforward manner by splitting the detector into distinct regions. This is a viable solution in certain cases and some contingency for extra parallelism has been built into the number of processor boards manufactured. However, this approach is ultimately limited as an upgrade path for Level 2, because of the finite numbers of processor boards available and also due to space limitations in the Level 2 crates.

Not all processing can benefit in direct proportion to the amount of parallelization in the hardware. This is the case where an algorithm must correlate information across an event. The Level 2 global worker may be duplicated so that each worker processes separate events or a single event's processing may be partially parallelized between workers. The advantages in the former case are limited by the need to report trigger decisions in the order in which events arrive ("worst

---

<sup>1</sup>In fact, given the rapid evolution of computer electronics, it would be difficult to complete a prototyping to production cycle without finding that various components have gone obsolete prior to final assembly.

of ‘n’ performance,” see Fig. 1) and in the latter case by Amdahl’s law [5]. In such cases direct gains in single CPU processing power are the most effective means to improve performance<sup>2</sup>.

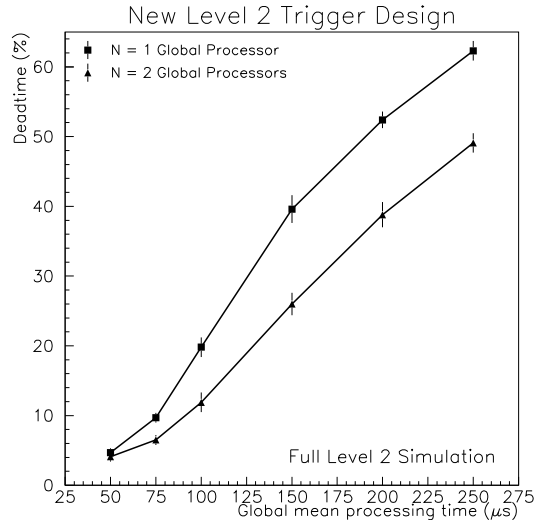


Figure 1: Timing results for a scenario with two Global Level 2 workers handling alternating events compared to a single worker system (from Level 2 Global TDR [5]). Since this is an  $N = 2$  serializing farm, a full factor of 2 gain is not attainable.

Comment on new uses for increased CPU. For example n-1,n,n+1 detangling of events - Kt, moving cone processing, etc.

### 3 The L2 $\beta$ Strategy

In order to buttress the long term performance and reliability of the DØ Level 2 trigger, we propose a processor system compatible with future performance upgrades and allowing for long term replacement of system components. Rather than attempt a redesign of the entire processor board, we instead prefer to implement a system using a more modular approach, taking full advantage of industry-standard components and upgrade paths. We follow the same conceptual design for the function of the board in our Level 2 system (moreover we require both hardware and software compatibility between both Alphas and  $\beta$ etas). However, this implementation replaces the CPU and assorted computer peripheral functions with a commercially produced single board computer (SBC). This SBC will reside on a 6U CPCI card providing access to a 64-bit, 33 MHz PCI bus via its rear edge connectors. Such cards are readily available “off the shelf” from numerous vendors (SBS-BIT3, VMIC, Concurrent Technology, etc., ...). The remaining functionality of the board will be implemented in a large FPGA and Universe II vme interface mounted on a 6U to 9U VME adapter card as shown in Fig. 2. The adapter card will contain all DØ-specific hardware for Magic Bus and trigger framework connections. The SBC, in the adapter, will sit will have its front panel

<sup>2</sup>Using another example taken from the Calorimeter Preprocessor, in the case of Level 2 jet clustering, greater CPU resources would allow for running a seedless (movable window) algorithm within the Level 2 time budget. This would offer the benefit of higher efficiency for low  $E_T$  jets even at lower luminosities and is a desirable feature for multi-jet triggers.

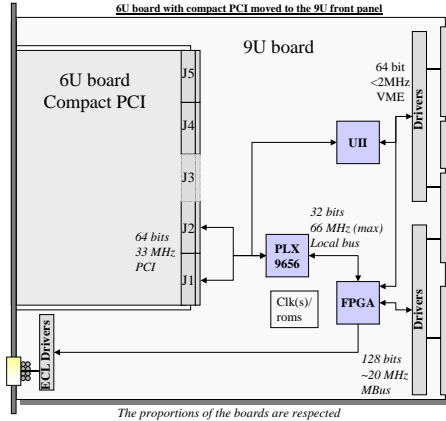


Figure 2: Physical model for the L2 $\beta$  processor card. Connectors J1 and J2 provide the 64-bit CPCI connection to the CPU. The functions available on J3-J5 may be assigned arbitrarily by each board manufacturer.

at the face of the crate, This implementation offers several clear advantages:

- The modular design, incorporating existing CPU cards, greatly reduces the engineering, debugging, and prototyping needed.
- The modular design provides a clear path for CPU performance upgrades by simple swapping of SBC cards.
- Compatibility allows for a phased-in replacement of the Alphas.
- A single slot design frees space in the trigger crates and allows greater flexibility for future SBC selections.
- Higher or lower performance CPUs may be selected. For example, lower performance, less expensive CPUs may be used in Administrator-type processor cards, while more powerful CPUs can be used where physics algorithms are run.
- The use of a much smaller number of components to implement the DØ-specific functions of the board greatly reduces the parts count, increasing reliability and ease of hardware debugging.

The last point above is of particular importance. The current implementation of the processor boards is composed of over 170 individual ICs mounted on a board composed of 10 printed circuit layers with approximately 9000 through-hole connections (vias) [7]. By contrast the custom-built components of the above implementation can be constructed with approximately 30 ICs (most of which are simple buffers or logic converters) on a vastly simplified circuit board layout. This is a direct result of using a manufactured CPU and VME interface module and greater integration of functions into a more modern FPGA <sup>3</sup>.

<sup>3</sup>Note: The only other cards in the Level 2 system that communicate via the MBus are the Magic Bus Transceiver (data broadcast) [8] cards. These cards already implement their primary MBus functions in a single FPGA. The addition of an interface with added programmability to the processor cards would also allow for future flexibility in the utilization of this bus.

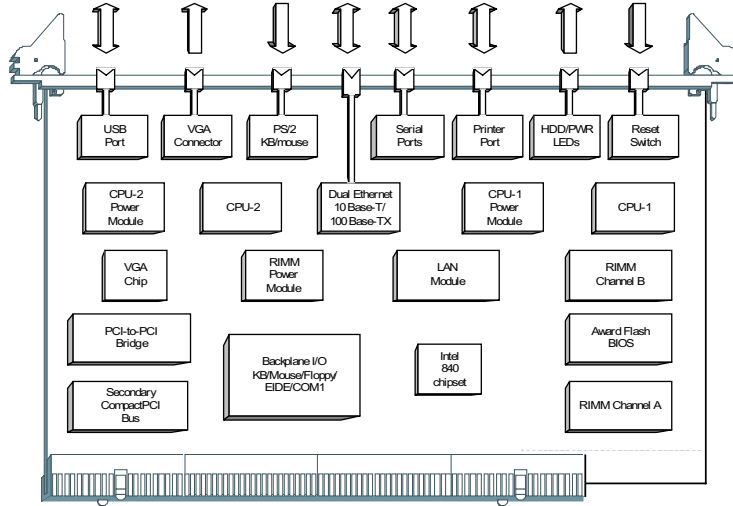


Figure 3: Advantech MIC3385 CompactPCI processor block diagram.

## 4 Hardware Overview

The  $L2\beta$ etas will be composed of two separate devices: a commercial single board computer with a 64-bit CPCI interface and a custom 6U-to-9U adapter card. Each components are described in the following sections.

### 4.1 $L2\beta$ eta SBC

Initially, we plan to use the MIC-3385 cPCI SBC [9] as our processor in the  $L2\beta$ etas. This board (and others in its class) can accept dual Pentium CPUs (up to 933MHz). Although we plan to initially order the cards with a single CPU installed, adding a second processor is very interesting as a inexpensive means to increase processing power. Competing manufacturers with very similar products include Diversified Technology Inc. [10], and Teknor [11]. We fully expect that additional manufacturers will be shipping boards with essentially the same features by the time the  $L2\beta$ etas go into production. Thus we have little worry of facing a single vendor situation for CPUs.

All CPCI boards mentioned above support a hard drive, CDROM, floppy, etc. through an EIDE controller accessed via manufacturer-specific pins on coenctors J3-J5. Our 9U adapter will have to intercept these signals and route them via a header to a hard drive mounted on the card. All cards provide integrated ethernet conections.

All of the CPCI cards are supported under Linux. And the KAI C++ compiler is readily available for this platform and OS. After inspecting PowerPC options in the CPCI SBC market, we have found that the software tools we require to implement the system in a straightforward way are not readily available for these choices. Furthermore, we expect the Pentuim flavor of Linux to more closely parallel our current Alpha version, thus reducing efforts in completing our necessary operating system modifications.

Table 1 compares the performance of various CPUs available today on CPCI SBCs to the Alpha PC164 benchmarks. The integer performance of the CPUs is of primary importance for most of the operations in  $D\emptyset$ 's trigger. Each modern CPU compares quite favorably in this category.

CPU Type	Specint95	Specfp95
Alpha 500 MHz	~ 15	~ 21
Pentium III 800 MHz	~ 38	~ 29
Pentium III 850 MHz	~ 41	~ 35
Pentium III 933 MHz	~ 45	~ 39
Pentium III 1000 MHz	~ 48	~ 41

Table 1: Performance comparisons [12] for several modern CPUs available on mass produced VME single board computers. The current Level 2 Alpha CPU is shown at the top of this table. The integer performance of the CPUs is of primary importance for most of the operations in DØ's trigger.

## 4.2 L2 $\beta$ eta 9U Adapter

The 9U card will both adapt the 6U SBC card to the 9U crate form factor and provide hardware for all custom I/O required of the processor cards. This includes all Magic Bus I/O, an interface to user-defined J2 lines, the VME interface and outputs to trigger scalars.

Details of the 9U adapter card are shown in Fig 2. All functions on this card will be implemented in a single FPGA plus assorted logic converters and drivers. The FPGA of choice is the Xilinx XCV405E [13]. This device is particularly suited to our application, because of its large amount of available Block RAM. 70 KB of RAM (in addition to > 10 K logic cells) is available to implement data FIFOs and Mapper (address translation counter) tables, thus greatly reducing the complexity of the 9U PCB. The replacment of various Alpha hardware components by the Xilinx 504E is further discussed in section 5.3.1.

## 5 System Requirements

This section summarizes the performance features and specifications of the L2Alpha processors and describes how the L2 $\beta$ etas will satisfy these requirements.

### 5.1 Processing Power

At the heart of the L2Alphas is a 500 MHz COMPAQ 21164 CPU supported by a PC164-style motherboard. The 21164 provides:

- 2 integer and 2 floating pipes (64-bits wide)
- 96 kB on-chip L2 cache (enough for event data + 12 K (8 byte) instructions of non-looping code execution (These characteristics **seem** to allow operation of the system with the (4 MB) L3 cache disabled.)
- **execution of proposed L2 algoritms in under 50  $\mu$ s/event**  
Performace benchmarks: Specint95 15, Specfp95 21

As shown in Table 1 the 850 MHz PIII compares very well to the ALPHA 21164 in terms of performance. It should also be noted that the PIII contains a larger on-die cache (256 KB versus 96 KB, this Level 2 cache runs at the clock rate of the CPU). The Pentium instruction set is more complex than the ALPHA (RISC processor) set. It contains command sets for backwards compatibility with less modern generations of Intel processors and numerous, complex instructions

dedicated to multimedia applications. Instruction sizes can range up to a maximum of about 16 bytes in size [6], however from reference [6], typical integer and branching instructions should be on the order of 8 bytes or less. Conservatively, with a 4KB event in its cache, the PIII should be able to hold approximately 32K instructions.

The PIII SBCs typically do not have L3 cache, however DØ already expects to run with the L3 cache disabled in the Alphas and it should be noted that the new PIII boards offer significantly faster RAM access than the alphas. The local RAM bus on the PIII's run at 100 to 133 MHz at 64 bits width. This speed is comparable to the PC164 L3 cache (although at half the data width).

## 5.2 Linux/RT Unix Operating System

Various versions of Linux are already in use on the Concurrent card and other PIII-based SBC models. The needs of the experiment require two modifications to standard Linux to provide real time behaviour and to allow direct mapping of physical memory within user programs. In order to provide real time behavior, a special interrupt handler must be implemented to sit below the Linux Kernel and to service interrupts, such as new event interrupts, in a timely manner.

Intel Linux offers the added advantage of the KAI compiler. The L2betas will in fact be more easily integrated into the DØ software environment than the Alphas. Intel Linux is fully supported at DØ and KAI is the compiler choice.

## 5.3 VME, TSI, MBus Devices

The Alphas contain four separate PCI devices to accommodate our special I/O needs: the UII VME interface; the TSI which handles a variety of MBus and J2 signals and data output to the trigger framework through a front panel ECL connection; and two Magic Bus data interfaces: A PIO FPGA and a CPLD responsible for initiating DMA transfers from FIFOs receiving MBus broadcasts.

The UII interface hardware will be identical on the L2betas. The remaining device functionality will be contained in the Xilinx FPGA and the PLX 9656. The functionality of the devices will be simplified from their Alpha counterparts, because they will not require firmware blocks to implement PCI compliant devices. Situating the functions in a single device, using a single programming tool reduces support and development overhead.

### 5.3.1 Hardware Requirements

The Alphas contain 128 bits  $\times$  4K of data FIFOs plus an address FIFO of the same length for MBus broadcast addresses. Broadcast addresses are restricted to the 10-bit MBus address range, [0:1023]. The Mapper contains address translation (MBus to PCI) information for each broadcast address. Its registers are incremented as the FIFO's are drained and at any given time provides the PCI target location for data at the head of the FIFO. The Mapper registers may be read to determine how many words were written to each broadcast address. The nominal data storage requirements for these devices is:

Data FIFO: 128 bits  $\times$  4 K = 64 KB  
Address FIFO: 10 bits  $\times$  4 K = 5 KB  
Mapper: 32 bits  $\times$  1 K = 4 KB

Total: 73 KB.

This results in a 3KB shortage of memory compared to the 70KB available in the Xilinx XCV504E. Xilinx produces their model XCV812E with twice the block RAM in the same package at the increased price of  $\sim$  \$1100 compared to  $\sim$  \$500 for the XCV405E. However, there are a number of other solutions to the shortfall.

Rather than dedicating a large number of logic elements to fill the 3KB shortfall, it is possible to FIFO only address changes. In this case we would dedicate 1 bit  $\times$  4K to create an address change flag FIFO (or 129<sup>th</sup> data bit). The remaining 7bits  $\times$  2K of block RAM could be used to construct a reduced address FIFO of 10 bits  $\times$  1 K.

Alternate solutions<sup>4</sup> include reducing the depth of the data FIFO capacity to 3.75 KB. This would require the addition of an explicit ‘hold off’ mechanism in the DMA protocol to prevent FIFO overflows. The DØ MBT cards are compatible with such a mechanism. This is accomplished by use of the DDONE handshake line on the MBus backplane.

Finally, given the limits on numbers of broadcast sources and strict ordering of source broadcasts within the DØ crates, is plausible to use a set of registers instead of an address FIFO. The implications of such an approach on firmware complexity and higher level software has yet to be explored.

### 5.3.2 Functional Requirements

Various requirements must be met by the MBus devices:

- The 9U board is required to send data to memory in the SBC at a rate of around 80-100 MB/s.

There are three possible bottlenecks to explore for this requirement: the FPGA/FIFO’s, the PCI interface, the PCI controller. All of these devices are capable of delivering data at the full 64-bit PCI bus bandwidth of 264 MB/s. The PCI controller is fully capable of delivering data throughput at the PCI limit and the bandwidth of the memory bus far exceeds PCI. (Note: The Alpha PCI controller has only delivered  $\sim$  80 – 120 MB/s of DMA throughput despite its 64-bit width, because its buffers frequently fill while sending data to memory.) Clocking the Xilinx at sufficient speeds to deliver this bandwidth poses no difficulty.

Following are some conservative effective bandwidth estimates. The relevant numbers for DMA performance in DØ can be thought of as follows: The maximum size for a data source is about 320 bytes (Data sources transmit sequentially to to build an event of up to 4 KB in size). Assuming about 300 ns of startup time for a DMA transaction due to local bus arbitration, it would take 2.9 $\mu$ s to put a source’s data into memory (an effective bandwidth of 110 MB/s). Additional overhead is incurred if a PIO transaction interrupts the DMA. The timescale for such interruptions is on the order of the event processing time. Assuming that the PIII’s are twice as fast the Alphas, we might expect an average processing time (very roughly) on the order of 20 $\mu$ s/event. With three worker cards in a crate, a message might be sent to the Administrator roughly every 6 $\mu$ s. Therefore it is unlikely that the DMA from an individual

---

<sup>4</sup>All solutions consider the allowable configurations of block RAM elements in the XCV405E. Block RAM elements may be configured in the following width and depth combinations [13]:

Width	Depth
1	4096
2	2048
4	1024
8	512
16	256

source would be interrupted more than once before it completes (the rate would be lower for Administrator to Worker messages<sup>5</sup>).

- Programmed I/O precedence over DMA. It must be possible for the DMA to be preempted under two conditions:
  1. FIFO's are being read out and the receiving board needs to send a PIO transaction to the Magic Bus. This would be accomplished by programming the PCI latency times in the PLX 9656. In accordance with PCI protocol, a bus master must relinquish bus ownership after its PCI latency times expires, if its GRANT has been removed by the PCI controller. This would be the case if a PIO transaction were pending from the CPU. The latency time is fully adjustable in by setting the appropriate register in the PLX 9656. This is a important matter, because latency in reporting a finished event is charged directly against the time budget for a Worker Processor. We would set the latency timer to allow no more than 2-3  $\mu$ s of delay.
  2. The second condition could occur if the FIFOs are being read out and the Administrator Processor initiates a PIO transaction over the Magic bus. In this case the PLX can be caused to halt DMA from the local bus side and to re-arbitrate the PCI bus to complete the incoming PIO transaction. After this transaction, the DMA transfer would be reinitiated.
- DMA destinations must be configurable by the CPU for all MBus broadcast addresses. The full functionality of the DMA Mapper as used in the Alphas can be implimented inside the Xilinx, maintaining the required functionality. The PLX fully supports writing data to targets with a large range of local addresses on the local bus.
- It must be possible to interrupt the PCI bus in accordance with events on the local bus (FIFOs empty, new event, etc). The PLX explicitly supports generation of interrupts from the local bus.
- Magic Bus PIO must be able to support a number of data modes: Master/Write, Master/Read, Slave/Write, Slave/Read. In our design this is a matter for firmware only, bi-directional drivers are provided to send or receive MBus data.
- Fast MBus arbitration ( $\sim$ 15ns). This is consistent with realizable gate delay times in the Xilinx FPGA.

## 5.4 VME Interface

Tundra's Universe II chip provides the VME interface to the L2 $\beta$  processors. The Universe chip supports up to 64 bit PCI and VME transfers including DMA transfers. The Universe II databook [2] should be referred to for the use of the chip.

The UII is defined as device 0 on the secondary PCI bus of the L2 $\beta$  card. This is accomplished by connecting PCI AD(16) to the UII's IDSEL input. The Universe II chip is connected to all 7 levels of VME interrupts. Following CPCI and PCI-PCI bridge recommendations, the UII's LINT(0) line is routed to PCI INTA (comment this should go through the FPGA for future MSI!). A VME controlled interrupt will be used to reset the Worker processors in the event that they are

---

<sup>5</sup>Interrupts due to VME transactions would happen at a much lower rate. Their PCI budget would be charged against the PCI bus bandwidth remaining after broadcast DMA.

stuck in a processing loop. The Alphas are also configured such that a VME RESET will also reset the alpha processor CPUs. This function is enabled by making use of the *PRST#* line on the J2 CPCI connector (pin C17). This is an open collector line that causes a system reset when it is pulled low. This line is driven by *UII\_LRST\**.

#### 5.4.1 Registers

PCI Configuration Registers for the Universe II are accessed by asserting its *IDSEL* line (AD16) during a configuration read or write. The PCI configuration registers 0x10 (and 0x14) define the base address for the Universe's 4Kbyte register space. These registers are accessed to determine the base address of the UII.

The Universe's 4Kbyte register space has 3 logical sections. The lowest 256 bytes of the register map to the PCI configuration space. The upper bytes map to the VME Configuration and Status Registers. The middle part of the space contains the Universe Device Specific Registers (UDSR). (The addresses defined by 0x10 and 0x14 both point to the same set of registers, but one does it in memory space and one in I/O space. Which is which depends on a power-up configuration. This choice has no impact on our device drive software.)

#### 5.4.2 Description of Operation

This section gives an overview of the operation modes of the Universe chip. The Universe II user manual should be referenced for a detailed description. The Universe supports read and write transactions originating from the VME or PCI side. It supports eight separate VME slave windows for VME target behavior and eight separate PCI slave windows the VME master behavior. For transactions originating on the VME side, the Universe acts a VME slave. If the VME address falls inside the address range defined for one of the 8 VME slave images, the Universe becomes the PCI bus master and initiates the appropriate transaction on the PCI bus. To see if the VME slave image is being addressed, the upper 16 bits of the 32 bit VME address are compared to base and bound addresses for that slave image. This means the slave image window will be at least 64 Kbytes wide, and can be up to 4 GBytes wide. (Note that slave images 0 and 4 use the upper 20 bits and thus have a 4 KByte resolution.) The corresponding 32-bit PCI address asserted combines the VME address with the address in a translation offset register. The lower 16 bits of the VME address are used for the lower 16 bits. The upper 16 bits are derived from a twos complement addition of the upper 16 VME bits with the 16 bits of the translation offset register. There are also control bits for each image that further define the transaction (address modifiers, data and address width, etc.). The transaction can be coupled in which case the PCI transaction must finish for the Universe to respond to the VME side; or it is posted for a write, or prefetched for a read in which case an intermediate FIFO is used and the Universe can acknowledge on the VME side before the PCI transaction occurs. The Universe acts in much the same way if the transaction originates on the PCI side. Again there are 8 PCI target images whose registers define address windows and translation offsets. These images also have 64 KByte resolution with the exception of images 0 and 4 which have 4 KByte resolution. However, the PCI side does not support prefetched reads. Furthermore, if the PCI window is defined to be in PCI I/O space, posted writes are not allowed either, and all transactions are coupled.

DMA transfers can be initiated from either the VME or the PCI side.

**UII POWER UP OPTIONS- follow settings in comercial VME boards?**

Device	Supply Voltage					Total Watts
	12 V	5 V	3.3 V	-5.2 V	-12 V	
L2Alpha	2 A	20 A	10 A	1.25 A	1 A	176 W
L2beta SBC	248 mA	14.5 A	3.16 A			86 W
L2beta 9U						$\approx 10$ W
Hard Drive		0.5 A				2.5 W
(Max)Power usage for L2beta processor						$\approx 100$ W

Table 2: Power consumption per processor component. SBC test conditions reflect maximum power usage for Advantech board (two CPUs under load and 512 MB RAM).

5 V connections	
P1	A32, B31, B32, C32
P0	A1, B1, C1, C2, D1, E1
P2	B1, B13, B32
3.3 V connections	
P1	D12, D14, D16, D18, D20, D22, D24, D26, D28, D30
12 V connections	
P1	C31
-12 V connections	
P1	A31
-5 V connections	
P0	A4, A5

Table 3: Power supply pins on DØVME Crate backplane. Pins shown in **bold** are used to power the L2beta processor cards.

## 5.5 Mechanical and Electrical

The SBC will sit inline with the 9U adapter card. A cutout in the 9U card will be reinforced with aluminum bars to stiffen the adapter against torques around the BGAs and rails to guide the SBC. IDE signals provided on the P3 connector of the SBC will be routed to a hard drive header on the adapter card.

Table 2 provides a rough estimate of the power requirements on the L2beta as compared the L2Alpha. The power and heat dissipation requirements are significantly reduced in the L2betas.

Power is supplied to the 9U card via the P0, P1, and P2 backplane connectors of the DØVME Crate [15]. Table 3 lists the available power pins on the backplane and those that are used to supply the L2beta cards.

## 6 Software

In order to minimize the demand on the experiment’s resources to bring the L2betas to completion it is vital that this project tread as lightly as possible on other trigger groups’ projects. We have planned at the onset that the L2betas are to be both hardware and software compatible with the Alphas. Hardware compatibility will largely be a product of proper firmware design. By software

compatibility we are referring to high level software. It should be possible to recompile online code with little or no changes to run on the L2 $\beta$ etas. This compatibility will be enforced by the hardware interface layer, or device driver software.

## 6.1 Operating system, compiler, build environment

The L2 $\beta$ eta processors will run Linux Redhat Version 7.1, using Kernel version 2.4.x. The OS version will remain fixed indefinitely for the duration of Run II, unless there are compelling reasons to upgrade (for example, incompatible upgrades in the D $\emptyset$ -supported machines for code building). We will use the KAI compiler and all code will be built using the standard D $\emptyset$  software environment.

### 6.1.1 Kernel Modifications

Excluding hardware device driver code, modifications to the Linux kernel include the *bigphysarea* patch and level2 memory module (loaded after boot). Below is a typical entry in *lilo.conf* to configure the *bigphysarea*:

```
# linux kernel w/ bigphysarea patch, reserving 16MB of data =4k*4k pages
image=/boot/bzImage-2-4-7-bigp
    label=linux-2-4-7bigp
    append="bigphysarea=4096"
    read-only
    root=/dev/hda5
```

## 6.2 Device Driver Software

The coding tasks for this project can be displayed as in Fig 4. The device driver code will require a rewrite of the hardware interface layer to communicate with the PCI bridge. The user interfaces for the device drivers will be unchanged from the present system. This is necessary for source code compatibility.

The basic structure of the device drive code is to start with a generic PCI device and to build specific functionality into classes derived from this device. All hardware interface functions are hidden in this lowest level base class.

The UII driver will require the least amount of change, since this device is replicated on the SBC. In fact, the current device drive code is expected to function “as is” provided the PCI base classes it depends on are replicated for the PIII boards.

The remaining device driver code will have to be completely redesigned, since three former PCI devices will now live inside one FPGA. Different functions of this FPGA will be treated as pseudo-PCI devices to maintain a software interface compatible with the Alphas.

## 7 Firmware

The firmware will be composed in loosely coupled blocks similar to those shown in Fig 4. The most complex blocks will be the local bus interface and the MBus I/O block. A more detailed summary of the firmware blocks and FPGA resources required is shown in Fig. 5.

The necessary L2 $\beta$ eta functions can be implimented using 337 I/O pins. This includes “utility pins” allocated for configuration settings and logic analyzer ‘spy’ channels for assistance in firmware debugging. An additional 66 I/O pins are available for additional functionality we may wish to add

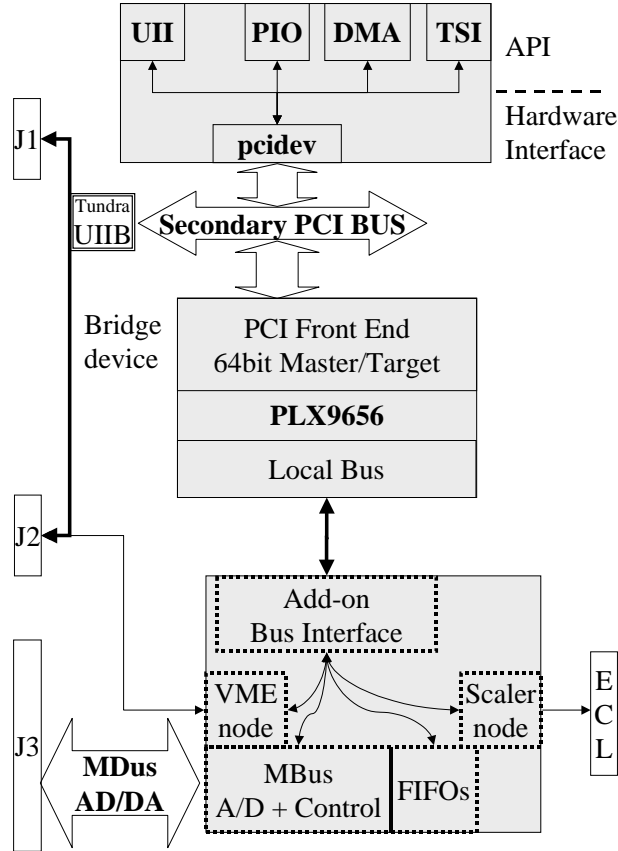


Figure 4: Block diagram of L2beta software and firmware components.

at a later time. The inclusion of the MBus data FIFOs in the FPGA has resulted in a great economy of I/O pins required.

## 7.1 PLX 9656 general configuration

The PLX 9656 provides a set of local configuration registers mapped to PCI Memory and I/O space and to a local bus address range. Additionally it provides four PCI memory windows for direct data transfers between PCI and local bus addresses. The general configuration for these windows is shown in Table 4.

The Control & Monitor Window provides access to all configuration, control, and status registers necessary for communication with the trigger system. This window is divided into four logical sections

Window #	Title	Size
0	Control & Monitor	64 KB
1	PIO Window A	64 KB
2	PIO Window B	64 KB
3	Reserved	

Table 4: PLX PCI memory window usage.

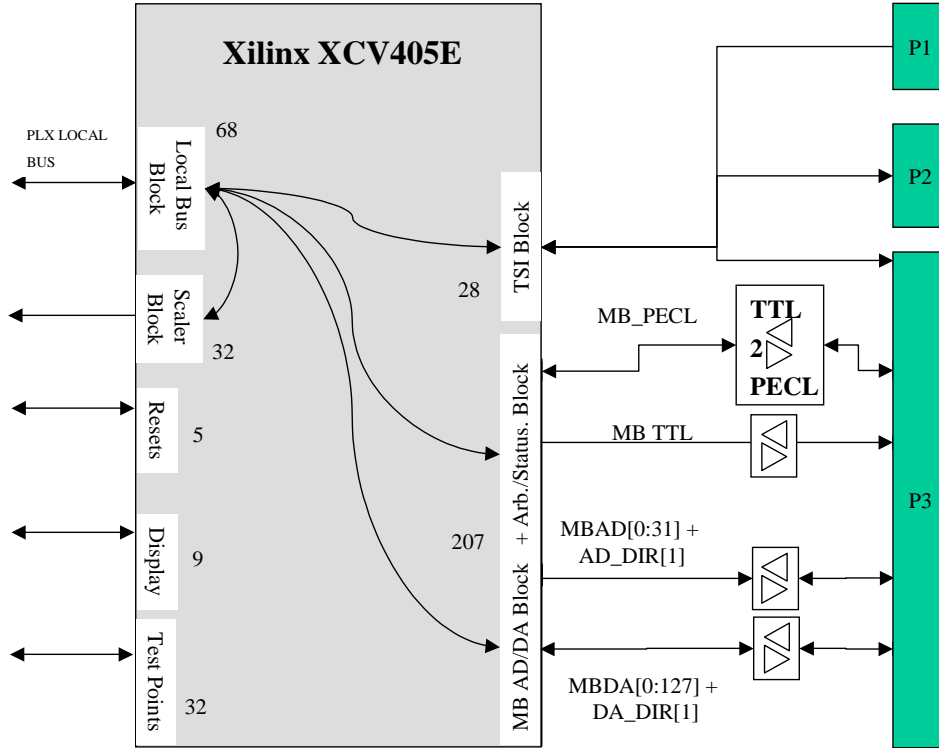


Figure 5: Overview of firmware blocks. I/O pin requirements are shown with each block. (UPDATE THIS FIGURE TO MATCH SCHEMATICS)

as shown in Table 5. The I/O Control window located at offset 0x0 is used to enable the DMA and PIO functions and to determine their operating modes.

## 7.2 Local Bus Firmware

In order to minimize development time and to maximize chances of success for the L2betas, our pilosophy has been to use commerical devices where custom devices can be replaced, to simplify the electrical design of the boards as much as possible, and to push as much of functionality as possible into firmware to reduce hardware prototype cycles.

Although we have sought to develop a system that is implemented in firmware to as large an extent as possible, we have choosen to use a hardware PCI interface rather than to complicate our firmware project with a PCI implementation. We plan to take full advantage of the PLX's features

Control & Monitor Window	
Offset	Function
0x0	I/O Control
0x10-0x1C	PIO Configuration
0x100-0x148	TSI Registers
0x1000-0x2000	Mapper Registers

Table 5: PLX PCI memory window 0 definition.

I/O Control Register		
Bits	Access	Function
0	R/W	enable DMA
1	R/W	broadcast lockout
2	W	clear fifo
7:3	R	0x0 (undefined)
8	R/W	enable PIO Target
9	R/W	PIO write to FIFO
31:10	R	0X0 (undefined)

Table 6: I/O control register definition

to simplify the firmware design.

Perhaps the most interesting feature of the PLX9656 is its ability to become a Master on both the PCI and local buses and to fully control DMA transactions (Figure 6 gives a brief overview). Thus we may simplify the DMA block of the firmware to a protocol that provides data words at the PLX's request. This removes all need for the Xilinx to directly support PCI timing, because the PLX completely decouples the two buses. The PLX can run its local bus faster than PCI (up to 66 MHz) to ensure that its internal buffers do not run out of data while feeding the PCI bus.

The PLX also offers a local bus protocol with separate address and data lines. Using this interface all transactions can be thought of as one step responses based on address decodes. This is conceptually easier to deal with than protocols that multiplex address and data lines where addresses and data states must be considered in the protocol.

A possible approach to the Local bus firmware is the following:

- Provide Local bus targets for PIO and Mapper registers, PIO writes (MBus master), PIO reads (MBus Master), and TSI (equivalent) writes/reads. Each of these Target would be specified by different local address.
- Provide Local bus masters for PIO writes (MBus Slave) and PIO reads (MBus Slave).
- Provide an Local bus master for interrupt generation. (Note: Pending local bus requests have higher priority than ongoing DMA, therefore the PLX protocol conveniently allows a mechanism for the local bus to halt DMA as needed.)
- Provide an Local bus Master to set up DMA registers in the PLX and initiate the transfer.
- Provide an Local bus target to feed data to the PLX's DMA engine.

It is likely that several (or all) of the Local masters would be combined into one module to reduce complexity of the Local bus operations.

### 7.3 MBus I/O and Arbitration Firmware

The MBus arbitration firmware block will implement the MBus arbitration logic. This essentially amounts to a series of combinatorial logic elements switching at speeds on the order of the (few ns) gate delays of the Xilinx.

The MBus I/O firmware (Fig. 7) is responsible for controlling the MBus Data and Address drivers and the (internal) Data FIFOs. It must respond to a requests from the PLX to send or



## 7.4 DMA Firmware Functions

The Magic Bus (MBUS) DMA interface provides a direct path for MBUS data to be placed into main memory. The lowest 1024 MBUS addresses (0:1023) are reserved for DMA broadcasts. A 4KB Translation Buffer (Mapper) is used to map each DMA address to a 32-bit memory address. In order to configure the Translation Buffer, the CPU must write the main memory address for each DMA channel into the TB. The TB buffer holds 1024 addresses which are accessed by the CPU by writing to the FPGA base memory address + 4(times) DMA number. The DMA number is 0 to 1023 and directly corresponds to the corresponding MBUS address. The PCI address is 4 times that since we are accessing 32 bits of data and a PCI memory transaction addresses at the byte level. For example, for base memory address 0x02000000 the TB buffers would be addressed by writing to 0x02000000, 0x02000004, 0x02000008, 0x0200000C, etc. The data written to the TB is the (PCI) memory address to which data will be sent.

### 7.4.1 Description of DMA operation

The MBT boards send their data to the processor boards over the Magic Bus by addressing one of the DMA channels in the 32-bit Magic Bus Address space. The DMA addresses are reserved to be the lowest 1024 addresses in MBUS address space. These correspond to MBUS Address bits 31:10 being set to zero.

Each MBT board will be assigned its DMA channel(s) to use during initialization. When any of the 1024 DMA addresses are asserted on the MBUS, the processor board will clock the 128 bits of MBUS data and lowest 10 bits of the MBUS address into a 4K deep FIFO. The  $\beta$  board responds with *DDONE\**, ending that MBUS transaction. An Alpha or  $\beta$  can receive data into its FIFOs independent of any action on the part of the rest of the DMA engine. If the FIFO fills up, no error is given, and subsequent data sent to the Alpha will be lost, although in practice this should never be a problem. Another approach would be to have the processor withhold *DDONE\** when its FIFO's are full. This would be necessary if event storage requirements exceeded 64KB or if smaller FIFOs were used.

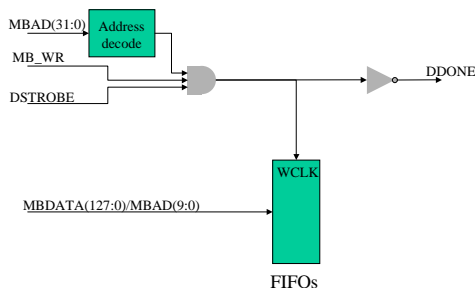


Figure 8: Broadcast address decode logic.

When the FIFO EMPTY flag goes false, the initiate a data to main memory by enabling one of the DMA engines in the PLX. This requires setting of both the local bus address to draw the data and the PCI address for the DMA target followed by a ‘go’ command. The Translation Buffer contains the starting 32-bit PCI address. The PLX arbitrates to gain control of both the local bus and the PCI bus and starts sending data as soon as it can. The DMA engine always performs

a 64-bit data transfer, and always sends at least two consecutive data words (= 1 MBus word) following one address.

While data providing data to the local bus, the address for the next data in the FIFO is compared with the previous address. If the address is the same, the data is sent as part of the same PCI burst transaction corresponding to the initial address that was sent. Otherwise the DMA is halted and the PCI target address is reset.

While the data is being sent, the PCI address in the Translation Buffer is incremented to point to the next PCI address for that DMA channel. The PCI address is incremented by 16 bytes for each MBWORD sent since each MBWORD is 16 bytes long<sup>6</sup>. When all the data is received for a given event, the Translation Buffer can be read to find the ending PCI address written into the Translation Buffer. The difference between the initial and final address in the Translation Buffer indicates the number of bytes that were transferred. The Translation Buffer must then be reconfigured to setup the addresses for the next event.

By turning off the appropriate bit in the I/O Control Register (Table 6), the DMA engine will not automatically try to send data from the FIFO to main memory. In that case the FIFOs would still receive data and eventually fill up, unless the Broadcast Lock-out bit is also set, disabling the FIFO from receiving data from the MBus.

The PCI latency timers in the PLX and PCI-PCI bridge should both be configured to allow for a halt in DMA after 1-2ms if a pending request exists on the main PCI bus. This is necessary to limit latency times for PIO output transactions from the CPU.

Additional comments:

- Discussion of the DMA firmware model suggests an ‘address change’ fifo as a means to fit all DMA functions into the limits of our Xilinx XCV405E fpga. For an initial design of the firmware, it may be easier to bring the system up while using a standard address FIFO and configuring the DDONE logic to halt the DMA from the MBTs when the FIFO’s fill. Given the speed of DMA transfers through the PLX chip, the performance penalty for this should be minimal.
- PIO precedence over DMA must be implemented. The case of PIO output transactions is described above. PIO input transactions are discussed in the next section.

## 7.5 PIO Firmware Functions

The Magic Bus Programmed I/O module is a bridge between the PCI and Magic Buses allowing reads or writes from one address space to the other. The PCI-MBUS transaction can originate from either side and can be either a read or a write. This allows the CPU to directly access the Magic Bus memory space or for a board on the Magic Bus to write or read into PCI memory space. The FPGA module supports up to 64-bit PCI data transfers but only 32-bit addressing, and accesses all 128 bits of MBUS data.

### 7.5.1 Configuration

The PIO module is configured by writing to the appropriate configuration registers mapped to Window 0 in the PLX. Access is accomplished by a memory write to the appropriate offset in this

---

<sup>6</sup>It should also be noted that only a 16-bit Adder is used to increment the PCI address bits (18:3) in the TB. As a result the upper 13 address bits are unchanged. That means that for a given event, each DMA channel cannot write past a 512 KByte boundary. Therefore the starting address used to configure the TB should not be too close to that boundary. This limitation could easily be removed on the betas, however we should take care to preserve compatibility with the Alpha processor online code.

PLX Control & Monitor Window #0 PIO Configuration Registers		
Offset	Access	Function
0x10	R/W	PCI Translation Base
0x14	R/W	MBUS Upper Memory Address
0x18	R/W	MBUS Lower Memory Address
0x1C	R/W	BUS Translation Base
0x20	R/W	Mbus Error Register

Table 7: PIO Configuration registers. Offset values are relative to the PCI Memory base for PLX window 0.

window. The PIO module uses 3 separate 16-bit (64K) memory address spaces mapped to three windows in the PLX. The first is for on chip registers that hold the PCI to MBUS translation address, as well as the MBUS address space reserved for the chip. The other two address spaces are PCI addresses defining Window A and Window B (PCI windows mapped to a Mbus address window). Table 7 lists the values of the PCI configuration space.

PIO configuration registers 14h and 18h must be configured with the PCI address for Window A and Window B. These windows are described below.

### 7.5.2 PCI Transactions

Addressing in PCI memory address space is accomplished by using the lines AD(31:2) to address a DWORD (4 bytes). The MAGICFPGA must convert this addressing scheme to the MBUS address, where the MBUS address lines (31:0) address a single 128 bit MBWORD (16 bytes). When the CPU initiates a read or write to the MBUS, it can choose one of two PCI address windows to address the MAGICFPGA. These are called Window A and Window B. Each of them is a 64K space in PCI memory address space and their base PCI address is set in the PCI configuration register as described below. Both windows still access the same MBUS address space which is fixed by the PCI Translation Base. The purpose of having separate windows is described in detail below. The upper 16 bits of the PCI Translation Base gives the upper 16 bits for converting a PCI address to a MBUS address. Since 16 bits are used for the translation, that means only 64K of contiguous MBUS address space can be addressed. In order to address a MBUS address that is not within 64K of the PCI Translation Base address, the PCI Translation Base register would have to be updated before attempting to read/write to that MBUS address. Figure 9 illustrates how a PCI address is converted to a MBUS address.

PCI to MBUS mapping a concrete example:

Assume the following configuration for the MBUS PIO registers: Base address of PIO setup registers: 1090000 Base address of Window A 1100000 PCI Translation Base 100000 (register offset 0x0 from 1090000)

Then a PIO write to Window A at PCI address 1100000 will generate a Mbus cycle with address 100000. A PIO write to Window A at PCI address 1100010 generates a Mbus cycle with address 100001. The reader should be reminded that the PCI bus is addressed by bytes and the Mbus is addressed by Mbus (16-byte) words.

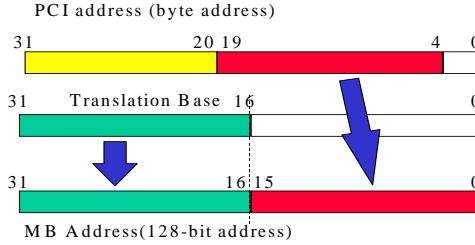


Figure 9: PCI address to MBUS translation

When the PCI side initiates a read or a write to the MBUS, the FPGA will try to gain control of the MBUS and then perform the transaction. The PCI and MBUS transactions are coupled so that the PCI transaction doesn't end until the MBUS transaction is finished. If the FPGA cannot gain control of the MBUS within 16 PCI cycles, the PLX should assert STOP on the PCI bus. This will signal the PCI controller to retry the transaction, so no software retry will be necessary. If the FPGA is simultaneously addressed from both the MBUS and PCI sides, the transaction originating on the MBUS will take precedence and the PCI transaction will fail.

In the beta firmware an appropriate timeout should be included, so that the PCI bus can be released if no target responds to the read/write request. An error register should also be added to reflect the state of the last read/write transaction on the MBus master.

The two PCI windows on the MAGICFPGA are accessed separately by two different PCI addresses. However, their purpose is not to address separate MBUS addresses but to allow for two different types of PCI transactions. PCI transactions can occur as 32-bit or 64-bit, can be burst or non-burst, and can be from sparse or dense memory space. Having the two windows allows some flexibility in having different types of transactions.

The basic difference in the two windows is that Window A is used for PCI data transactions of less than 128 bits, while Window B always requires a PCI transfer of 128 bits. In either case, the Magic Bus transaction is a full 128 bits, but all data bits written may not all be meaningful for Window A. The distinction between Window A and Window B is related to the distinction between PCI sparse memory space and PCI dense memory space.

When writing to Window A, the MBUS transaction starts when the last PCI data transfer occurs. (The last data transfer is indicated by the PCI protocol.) The data can be just 32 bits or up to 128 bits in a burst transaction, although the burst should not go past 128 bits.

Window B only initiates a MBUS transaction after a 128 bit transfer has occurred, which can be done via a burst of four 32-bit transfers or a burst of two 64-bit transfers. Window B initiates a 128-bit transfer whenever the upper 32-bits of a MBus word are written. For example, if Window B sits at PCI memory address 0x1200000, the a transaction begins when there is a PCI write to address 0x120000C, 0x120001C, 0x120002c, etc When the write to 0x120000C occurs, the MBus word written corresponds to the data at registers: 0x1200000-0x120000C.

A read into either Window A or Window B will trigger a 128-bit read on the Magic Bus. The FPGA should support a burst read of up to 128 bits using either 32 or 64 bit transfers as well as individual 32-bit reads.

### 7.5.3 Magic Bus Transactions

A transaction on the processor card originates from the MBus side when an address asserted on the MBUS falls between the MBus Upper Memory Address and the MBus Lower Memory Address. These addresses must be configured in a register on the FPGA as described below. This causes the FPGA to ask for control of the PCI/Local bus and it will remain in that state until control is given. After receiving control, it initiates a 64-bit PCI transaction and will perform two 64-bit reads/writes in a PCI burst transaction. The FPGA will not end the MBUS transaction with the DDONE\* signal until the PCI/Local transaction is finished.

The MBus Upper Address and MBUS Lower Address define the window in MBUS address space for which the  $\beta$ eta board will respond to a MBus address. Only the upper 16 bits are used for the comparison so the smallest window defined is 64K. The MBUS Translation Base defines the base PCI address used for converting a MBus address to a PCI address. In this case the upper 12 bits (31:20) define the base PCI address. Therefore only 1 MByte of contiguous PCI memory space can be addressed. Figure 10 illustrates how a Magic Bus address is converted to a PCI address.

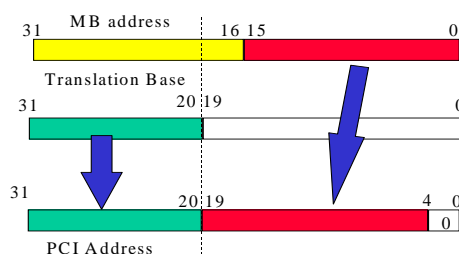


Figure 10: MBUS address to PCI translation

MBus to PCI mapping a concrete example:

Assume the following configuration for the MBUS PIO registers: RAM base: Z0000000 (This setup is specific to the PCI controller. Its function is to map PCI addresses to physical memory addresses. In our case any access to PCI addresses between 0xZ0000000 and 0xZ0000000+128MB will directly access the onboard RAM.) MBus upper address: 0x110000 (register offset 0x4 from 0x1090000) MBus lower address: 0x100000 (register offset 0x8 from 0x1090000) MBus Translation Base: 0xZ1000000 (register offset 0xC from 0x1090000)

Then a PIO transaction at MBus address 100000 will generate a PCI cycle on the MBus target at PCI address 0xZ1000000. A PIO transaction at MBus address 100001 generates a PCI cycle with address Z1000010.

As stated above, PIO transactions are considered to be fully coupled transactions (an important issue raised by PCI standards is discussed in section 7.5.4). Therefore, the PCI bus is held during the full MBus cycle. From the PCI bus the transactions may be described as follows.

MBus write cycle (I: Initiating processor, T: Target Processor)

1. I: PCI controller sends data to PIO device, PCI bus is held awaiting TRDY\*

2. I: PIO device gains control of MBUS
3. I: PIO device sends address/data to MBUS
4. T: PIO device latches data
5. T: PIO device gains control of Targets PCI bus and transfers data (4x32 bits or 2x64bits)
6. T: PIO device sends DDONE\* and releases PCI BUS
7. I: PIO device sends TRDY\*, PCI controller releases PCI bus

Mbus read cycle (I: Initiating processor, T: Target Processor)

1. I: PCI controller initiates read transaction from PIO device
2. I: PIO device gains control of MBUS
3. I: PIO device sends address to MBUS and waits for data
4. T: PIO device gains control of PCI bus, initiates a read transaction (4x32 bits)
5. T: PIO device places data on MBUS, sends DDONE and releases PCI bus
6. I: PIO latches data and returns a 32 bit work to the PCI controller

Comment: MBUS read transactions are initiated when reading addresses that are 128-bit aligned. Therefore to read a 128-bit word into the CPU the transaction could be done in two parts (four parts if 32-bit reads are used): 1) read a 64-bit word, aligned with a 128-bit boundary, from the PIO device - this initiates a MBUS read. 2) read the 2nd 64-bit word from the PIO device - this read need not initiate a transfer, because the data are already in PIO device's buffer.

In the beta design, it is entirely possible to allow posted (or burst) writes in Magic Bus PIO with additions to the firmware. This should be considered for a later stage in the project.

#### **7.5.4 PIO and PCI Compliance**

PCI specifications require that targets complete their first data transaction within 16 PCI clock cycles from the assertion of FRAME#. Care must be taken to buffer read requests from the MBUS within the Xilinx to provide compatibility with the PCI 'retry' mechanism. See ?? for details.

#### **7.5.5 DMA vs. PIO Priorities**

We require that PIO take precedence over DMA. The strict requirement is that delays for PIO transactions should not be large due to waiting for DMA transactions to finish. DMA must halt in response to a request for the PCI bus by the PCI controller. As mentioned in the previous section, this is accomplished with PCI latency timers. Additionally, the PIO target firmware must have the ability to halt DMA in order to finish a pending transaction. This may be accomplished on the betas by use of control lines on the local bus of the PLX9656.

I/O	TSI Connection ID	J2 PIN	Administator Function	Worker Function
in	Ext Test Interrupt	A24	SCL Init Interrupt	Worker Interrupt
in	L2 Answer Ready	A23	L2 Answer Ready	VME Busy
in	VBD Done	C25	VBD DONE	MBUS Busy
out	VBD Start Request	A21	VBD Start	Test Out
out	J2 Test Output 0	C21	Worker 1 Interrupt	
out	J2 Test Output 1	C22	Worker 2 Interrupt	
out	J2 Test Output 2	C24	Worker 3 Interrupt	
out	J2 Test Output 3	C32	Worker 4 Interrupt	

Table 8: J2 Trigger Connections.

## 7.6 TSI Firmware Functions

The TSI interface fully implemented in the Xilinx FPGA. This interface only functions as a PCI slave and is used to receive and send information to the rest of the trigger system. This includes the DØ trigger signals from the P2 backplane, as well as direct communication with the Trigger Control Computer (TCC) through a front panel connector. It is also used for monitoring the status lines of the Magic Bus.

### 7.6.1 Description of Operation

Because the device is a PCI slave, the only PCI operations are reads and writes to registers within the FPGA. Once the device has been configured with its base PCI memory address, one reads or writes to specific registers using the base address+local address. This device only supports 32 bit PCI transactions. The device has one of its pins connected to the PCI interrupt PAL at pin 9.

### 7.6.2 DØ Trigger Signals

The TSIFPGA is connected to 37 CDF trigger signals on the P2 backplane, however the DØ flavor of the L2Alphas uses only a small subset of the P2 connections as shown in Table 8.

### 7.6.3 Communication with Global Level 2

The communication with the trigger framework is through a 68-pin front panel connector. We send 32 signals of differential ECL, connected on board to 32 TTL signals connected to the TSIFPGA. The value of all 32 bits can be read/written from a single PCI register.

### 7.6.4 Monitoring of Magic Bus Status Lines

The TSIFPGA also monitors the Magic Bus status lines. These are the MOD\_DONE(18:0), EV\_LOADED(4:0), and MB\_AP\_FIFO\_EMPTY lines. Reading the appropriate register directly reads the value of these signals from the backplane. The empty status of the on board DMA FIFO is also read from this register. Writing to another register will drive the START\_LOAD, BUFFER(1:0), MBRESET, and AP\_FIFO\_EMPTY lines. This AP\_FIFO\_EMPTY line is connected to an open collector driver. The other 4 lines are connected to a transceiver. Only one of the Alpha boards in each crate will be the Magic Bus crate master and drive these lines. Writing a 1 to the CRATE\_MAS\_DIR bit will appropriately set the transceiver direction to drive these lines

on the backplane. The other boards should then have a 0 written to that register bit, in which case the value of these lines can only be read. (see Alpha schematic, sheet 40 - SEE APPENDIX). The TSI may also be software enabled to generate an interrupt under either of two conditions:

1. New event Interrupt (Fig.11):  
 (MOD\_DONE MASK is satisfied) .AND. (LOCAL FIFO EMPTY .OR. AP FIFO EMPTY)  
 .AND. (Interrupt 1 enabled)
2. SCL Initialize (Fig.12):  
 (SCL Initialize line on J2 pulled high) .AND. (Interrupt 2 enabled)

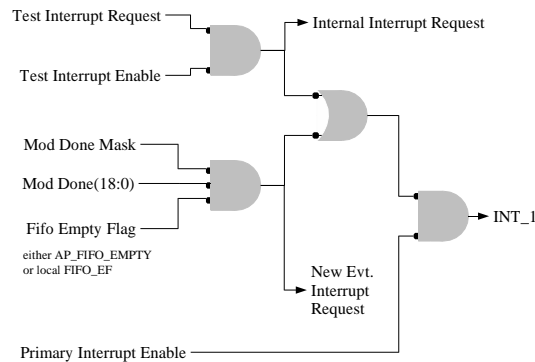


Figure 11: Interrupt 1: New event interrupt logic.

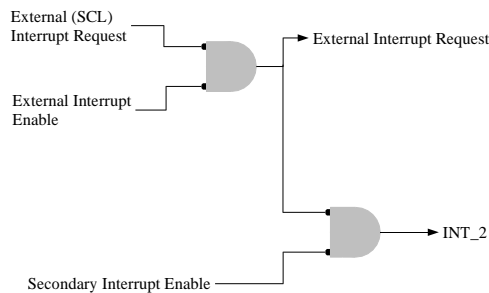


Figure 12: Interrupt 2: SCL initialize interrupt logic.

### 7.6.5 TSI Memory Registers

The memory location of the TSIFPGA registers and the signals accessed by each bit are shown in Tables: X-Y. The register addresses represent offsets from the PCI memory base address for the TSI register set. Each register bit is listed as an input or an output. Input bits monitor that status

Address offset	Access	Bits	Description	Connection/ Comment	I/O: Level
Broadcast Status Register					
0x10C	R	18:0	MOD_DONE(18:0)	J3 C41-B45	I: TTL
		19	FIFO_EF	see Fig. 14 Local fifo empty	internal
		20	AP_FIFO_EMPTY	J3 B40	I: TTL
		24:21	EV_LOADED(3:0)	J3 A46-D46	I: TTL
		25	MBRESET*	J3 B38	I: TTL
		26	Buffer(0)	J3 B41	I: TTL
		27	Buffer(1)	J3 A41	I: TTL
		31:24	0x0		

Table 9: TSI functions register map: Register 0x0C.

of bus signals. Output bits drive bus signals. For each output bit an internal register is written to control drives on the bus line. Reading back a register bit that drives an output provides the REGISTER value and not the bus level directly (see Fig. 13).

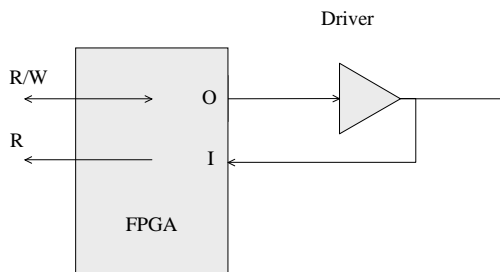


Figure 13: Crate master driver logic. Driving and monitoring of output lines. Two FPGA registers bits are used in these cases, one to drive the bus and one to monitor the bus. In some cases the output driver may only be enabled if a CRATE\_MASTER\_BIT is set on the board.

At present no module in the DØ L2 trigger drive the lines Buffer(1:0). These are fully available for future uses as the trigger evolves.

### 7.6.6 Summary of new i/o functionality

In addition to replicating all of the Alpha's functionality, the L2 $\beta$  processor will provide the following added I/O features:

- Buffer(1:0) lines are available for I/O on MBUS.
- Four J2 crate master bits are reserved for sending J2 interrupts to workers.
- Eight J2 inputs and eight outputs are reserved for any future need.

Address offset	Access	Bits	Description	Connection/ Comment	I/O: Level
Crate Master Register					
0x110	R/W	0:1	(undefined)		
	R/W	2	CRATE_MASTER	local control line	
	R/W	3	(TSL_)DONE_OUT	J3 D45	O: TTL64
				Bits (7:4) only driven when Crate Master is set	
	R/W	4	BUFFER(0)	J3 B41	O: TTL64
	R/W	5	BUFFER(1)	J3 A41	O: TTL64
	R/W	6	START_LOAD*	J3 C40	O: TTL64
	R/W	7	MBRESET*	J3 B38	O: TTL64
	R/W	8	(undefined)		
	R/W	9	VBD_START_REQ	J2 A21	O: TTL
	R/W	15:10	(undefind)		
	R	16	VBD_DONE	J2 C25	I: TTL
	R	17	0x0	(undefined)	
	R	18	L2_ANSWER_READY	J2 A23	I: TTL
	R	31:19	0x0	(undefined)	
Scaler Register					
0x114	R/W	31:0	TSL_OUT(31:0)	32 ECL out lines	O: ECL

Table 10: TSI functions register map: Registers 0x10 and 0x14.

Address offset	Access	Bits	Description	Connection/ Comment	I/O: Level
Internal Control Register					
0x130	R/W	18:0	MOD_DONE_MASK(18:0)		
	R/W	19	Select FIFO_EMPTY_FLAG	0: AP_FIFO_EMPTY 1: Local FIFO_EF	
	R/W	20	New Event Interrupt Enable	0: Disabled 1: Enabled	
	R/W	21	Internal Test Interrupt Enable	0: Disabled 1: Enabled	
	R/W	22	External (SCL) Interrupt Enable	0: Disabled 1: Enabled	
	R/W	29:23	(undefined)		
	R/W	30	Primary Interrupt Enable	0: Disabled 1: Enabled	
	R/W	31	Secondary Interrupt Enable	0: Disabled 1: Enabled	

Table 11: TSI functions register map: Register 0x30.

Internal Request Register					
Address offset	Access	Bits	Description	Connection/ Comment	I/O: Level
0x134	R	0	Internal Interrupt Request	Test_INT_Req.	
	R	1	External Interrupt Request	& Internal_Test_INT_Enable External Interrupt Req. & External Int. Enable	
	R	2	New Event Interrupt Request	(MOD_DONE & MOD_DONE MASK) & FIFO_EMPTY_FLAG & NEW_EVT_INT_ENABLE	
	R	7:3	0x0	(undefined)	
	R	8	External (SCL) Interrupt Req.	J2 A24	I: TTL
	R	29:9	0x0	(undefined)	
	R	30	INT_1	0: Deasserted 1: Asserted	
	R	31	INT_2	0: Deasserted 1: Asserted	

Table 12: TSI functions register map: Register 0x34.

Address offset	Access	Bits	Description	Connection/ Comment	I/O: Level
Internal Test Register					
0x13C	R/W	0	Test_Int._Req	1 = Request	
	R/W	4:1	J2_Crate_Mas. Output(3:0)	J2 C32, C24, C22, C31	O: TTL
		31:5	(undefined)		
User Output Register					
0x140	R/W	7:0	Additional J2 Outputs	J2 A(15:12) A(10:9) A(7:6)	O: TTL
User Input Register					
0x144	R	7:0	Additional J2 Inputs	J2 A29,A25,A22 C(11:7)	I: TTL
Geographic Address Register					
0x148	R	4:0	GA(4:0)	J1 D10,11,12,13,15,17	I: TTL
	R	5	GAP	J1 D9	I: TTL
	R	31:6	0x0	(undefined)	

Table 13: TSI functions register map: Registers 0x3C-48.

- Since the ECL outputs are driven from the Xilinx, they may be configured a firmware sopy channels is the need arises.

## 7.7 VME and Scaler Firmware

The VME and scaler firmware block is NOT required to provide any VME functionality. It will provide the access to User J2 lines and VME GEO lines as does the TSI device in the Alphas. The scalar block will simply drive front panel ECL drivers.

## 7.8 Spy and Configuration Channels

Thirty-two channels are reserved to drive a header for the purpose of examining signals internal to the FPGA. Signals would be selected by firmware settings and used for debugging purposes. Other pins are provided to connect status LEDs and a configuration switch to control firmware functions.

# 8 Future Upgrade Paths

## 9 Acknowledgements

We gratefully acknowledge the keen eye and assistance of Jim Linnemann in our pursuit of the better  $\beta$ eta.

## A Magic Bus Connections and Arbitration

## B Document History

version 2.0.1

Add fix to ALpha's MBus arbitration logic to Appendix.

Remove remaining Alpha-specific text from PIO section.

version 2.0.0

First version of cPCI TDR. Separate schedule/management sections, significant additions to firmware sections. From this version on the document gravitates towards a detailed technical manual as opposed to an overall review. version 1.4

Update schedule: move forward date to order SBCs for prototypes (VMIC has 60 day lead time), adjust prototype production schedule to Orsay's shop schedule; minor cost updates; typo fixes.

## References

- [1] Nucl.Instr.Meth.A269:51,1988.
- [2] Tundra Semiconductor Corp., [www.tundra.com](http://www.tundra.com).
- [3] AlphaPC164 Technical Reference Manual,  
<http://ftp.digital.com/pub/Digital/info/semiconductor/literature/archives.html>
- [4] Level-2 Calorimeter Preprocessor Technical Design Report,

- [5] Technical Design Report for the Level 2 Global Processor, DØNote #3402.
- [6] Intel Architecture Software Developer's Manual, <ftp://download.intel.com/design/mobile/MANUALS/24319101.PDF>
- [7] <http://umwnt1.physics.lsa.umich.edu/alpha/schematics.htm>
- [8] DØ MBT pages: <http://macdrew.physics.umd.edu/dzero/trigger/mbt.html>
- [9] <http://www.Advantech-nc.com>
- [10] <http://www.dtims.com>
- [11] <http://www.teknor.com>  
  
[http://galileo.phys.virginia.edu/~rjh2j/l2beta/beta\\_docs/PCI/pmc\\_draft.pdf](http://galileo.phys.virginia.edu/~rjh2j/l2beta/beta_docs/PCI/pmc_draft.pdf)
- [12] Information on Spec measurements can be found at [www.specbench.org](http://www.specbench.org)
- [13] Xilinx XCV405E, [www.xilinx.com](http://www.xilinx.com), goto VIRTEX-EM products.
- [14] <http://www2.plxtech.com>
- [15] For full crate specifications see: <http://>
- [16] T. Shanley & D. Anderson, PCI System Architecture, 4th edition, Mindshare, inc. 1999.

PIN	A	B	C	D	E
1	GND	GND	GND	GND	GND
2	MBDATA(0)	MBDATA(1)	MBDATA(2)	MBDATA(3)	MBDATA(4)
3	MBDATA(5)	MBDATA(6)	MBDATA(7)	MBDATA(8)	MBDATA(9)
4	MBDATA(10)	MBDATA(11)	MBDATA(12)	MBDATA(13)	MBDATA(14)
5	MBDATA(15)	MBDATA(16)	MBDATA(17)	MBDATA(18)	MBDATA(19)
6	MBDATA(20)	MBDATA(21)	GND	MBDATA(22)	MBDATA(23)
7	MBDATA(24)	MBDATA(25)	MBDATA(26)	MBDATA(27)	MBDATA(28)
8	MBDATA(29)	MBDATA(30)	MBDATA(31)	MBDATA(32)	MBDATA(33)
9	MBDATA(34)	MBDATA(35)	MBDATA(36)	MBDATA(37)	MBDATA(38)
10	MBDATA(39)	MBDATA(40)	MBDATA(41)	MBDATA(42)	MBDATA(43)
11	MBDATA(44)	MBDATA(45)	GND	MBDATA(46)	MBDATA(47)
12	MBDATA(48)	MBDATA(49)	MBDATA(50)	MBDATA(51)	MBDATA(52)
13	MBDATA(53)	MBDATA(54)	MBDATA(55)	MBDATA(56)	MBDATA(57)
14	MBDATA(58)	MBDATA(59)	MBDATA(60)	MBDATA(61)	MBDATA(62)
15	MBDATA(63)	MBDATA(64)	MBDATA(65)	MBDATA(66)	MBDATA(67)
16	MBDATA(68)	MBDATA(69)	GND	MBDATA(70)	MBDATA(71)
17	MBDATA(72)	MBDATA(73)	MBDATA(74)	MBDATA(75)	MBDATA(76)
18	MBDATA(77)	MBDATA(78)	MBDATA(79)	MBDATA(80)	MBDATA(81)
19	MBDATA(82)	MBDATA(83)	MBDATA(84)	MBDATA(85)	MBDATA(86)
20	MBDATA(87)	MBDATA(88)	MBDATA(89)	MBDATA(90)	MBDATA(91)
21	MBDATA(92)	MBDATA(93)	GND	MBDATA(94)	MBDATA(95)
22	MBDATA(96)	MBDATA(97)	MBDATA(98)	MBDATA(99)	MBDATA(100)
23	MBDATA(101)	MBDATA(102)	MBDATA(103)	MBDATA(104)	MBDATA(105)
24	MBDATA(106)	MBDATA(107)	MBDATA(108)	MBDATA(109)	MBDATA(110)
25	MBDATA(111)	MBDATA(112)	MBDATA(113)	MBDATA(114)	MBDATA(115)
26	MBDATA(116)	MBDATA(117)	GND	MBDATA(118)	MBDATA(119)
27	MBDATA(120)	MBDATA(121)	MBDATA(122)	MBDATA(123)	MBDATA(124)
28	MBDATA(125)	MBDATA(126)	MBDATA(127)	MBAD(0)	MBAD(1)
29	MBAD(2)	MBAD(3)	MBAD(4)	MBAD(5)	MBAD(6)
30	MBAD(7)	MBAD(8)	MBAD(9)	MBAD(10)	MBAD(11)
31	MBAD(12)	MBAD(13)	GND	MBAD(14)	MBAD(15)
32	MBAD(16)	MBAD(17)	MBAD(18)	MBAD(19)	MBAD(20)
33	MBAD(21)	MBAD(22)	MBAD(23)	MBAD(24)	MBAD(25)
34					
35					
36					
37	MBAD(26)	MBAD(27)	MBAD(28)	MBAD(29)	MBAD(30)
38	MBAD(31)	MBRESET	RD/WR*	DSTROBE*	DDONE*
39	Reserved	BOSS	GND	BOSSREQ	BOSSGROUT
40	Reserved	AP_FIFO_EMPTY	START_LOAD*	Reserved	BOSSGRIN
41	BUFFER(1)	BUFFER(0)	MOD_DONE(0)	MOD_DONE(1)	MOD_DONE(2)
42	MOD_DONE(3)	MOD_DONE(4)	MOD_DONE(5)	MOD_DONE(6)	MOD_DONE(7)
43	MOD_DONE(8)	MOD_DONE(9)	MOD_DONE(10)	MOD_DONE(11)	MOD_DONE(12)
44	MOD_DONE(13)	MOD_DONE(14)	GND	MOD_DONE(15)	MOD_DONE(16)
45	MOD_DONE(17)	MOD_DONE(18)	Reserved	DONE_OUT	Reserved
46	EV_LOADED(0)	EV_LOADED(1)	EV_LOADED(2)	EV_LOADED(3)	Reserved
47	GND	GND	GND	GND	GND
PIN	A	B	C	D	E

Figure 14: MagicBus connections.

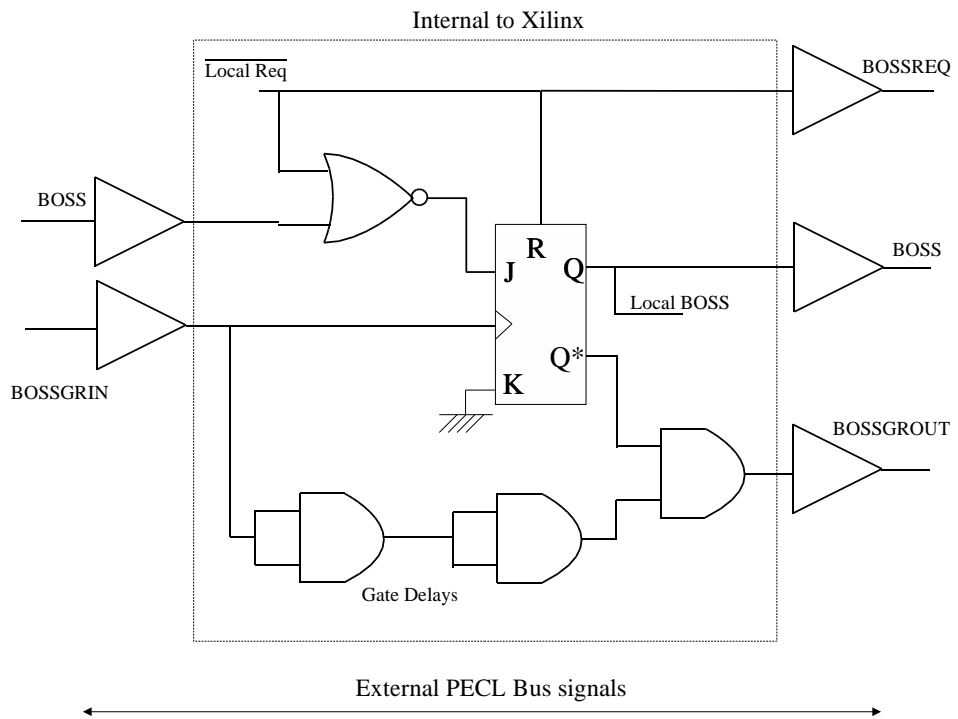


Figure 15: Boss logic for MBus arbitration.