Level 2 Processor Board PCI Device and FPGA Functions

6/27/2001

Bob Hirosky

The most current version of this note may be found at the L2βeta web site:
http://galileo.phys.virginia.edu/~rjh2j/l2beta#specs

This document is intended to serve as transitional source of information from the L2Alpha hardware specifications and descriptions. L2βeta TDR versions 2.0 and above will replace this document as the primary description for all facets of the L2βeta implementation.

1 General Description

This document describes the functions of the 4 custom PCI[1] devices implemented on the L2 Alpha processors. These devices are a Tundra Universe II VME interface, a Magic Bus to PCI **DMA** interface, a Magic Bus to PCI programmed I/O (**PIO**) interface, and a Trigger Supervisor interface (**TSI**). This document is intended to be a guide to the designs for L2βeta hardware and firmware. The ALPHA's functions and physical connections should be described in sufficient detail to replicate its functionality. Comments relevant to the L2βeta design are included in each section.

This document assumes that the reader has some basic familiarity with Magic Bus specifications, the Tundra Universe chip, and the PCI specification. Parts of this document borrow from the L2 Alpha Technical Document of S. Miller and Zhihui Huang.

2 VME Interface

Tundra's Universe II chip provides the VME interface to the Level 2 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. On the Level 2 Alpha boards the Universe II chip is connected to all 7 levels of VME interrupts. Interrupt line LINT(0) of the Universe chip is connected to pin 31 of the Alpha board's interrupt PAL. This PAL register corresponds to bit 2 of ISA address 804. See page 4-15 of the PC 164 TRM[3] for details. (The details are specific to the L2Alpha architecture only.)

---

[1] The 3 custom PCI devices implemented in FPGAs on the L2 ALPHA board may not be fully PCI compliant. This is especially true in the area of error handling.
[2] Available at http://galileo.phys.virginia.edu/~rjh2j/l2beta/#alpha_docs.
[3] ibid.

Following CPCI and PCI-PCI bridge recommendations[4], the UII's LINT(0) line would be routed to PCI INTA or INTB on the L2βeta. (We choose INTA by convention on the L2βeta.) A VME controlled interrupt will be used to reset the Worker processors in the event that they are stuck in a processing loop. The Alphas are also configured such that a VME RESET will also reset the alpha processor CPUs. Is it possible to retain this functionality on the βetas by making use of the PRST# line on the P2 CPCI connector (pin C17). This is an open collector line that causes a system reset when it is pulled low. This line should be driven by the UII_LRST* line with appropriate interface logic. (see Alpha schematics, sheet 16)

2.1 Register Configuration

PCI Configuration Registers for the Universe II are accessed on the Alpha by asserting its IDSEL line (IDSEL 16 on the L2 Alphas) during a configuration read or write. The PCI configuration registers 0x10 (and 0x14) define the base address for the Universe's 4Kbyte register space. In the L2βeta processors these registers will be accessed to determine the base address of the UII, exactly as in the Alphas. The UII's IDSEL line will be connected to PCI AD(16), making it device 0 on the subordinate PCI bus[4].

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.)

2.2 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

[4] See, for example, PCI system architecture / Mindshare, Inc.; by Anderson & Shanley.

derived from a two's 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.

2.3 UII Power up options

The Universe II may be configured at power up to take on certain default settings. On the L2βeta boards it should be possible to use default settings for all power up options, since these device configurations may be fully initialized in the device driver code.

2.4 Summary of UII issues for L2βeta

- Freedom to use fully default power-up configuration?
- UII interrupt line connects to INTA (and FPGA on beta).
- UII IDSEL connects to AD(16).
- Implement VME reset to CPU reset through PRST# on CPCI connector.

3 Magic Bus DMA Interface

The Magic Bus (MBUS) DMA interface provides a direct path for MBUS data to be placed into main memory. On the Alpha processors the PCI device is actually implemented in two separate programmable chips. A Xilinx XC4006E-2PQ160c is a (32 bit) PCI target (slave) that is accessed to write to the MBUS to PCI address translation buffers (Mapper). These buffers are a set of four 2Kx8bit static RAM chips (only half of this memory, 4 KB, is used). A Cypress CY7C372-125JC CPLD is used as the PCI master to write the data from the input FIFO to main memory.

3.1 Configuration of system

The Alpha's FPGA is configured by writing to the appropriate configuration registers according to the PCI specification. The FPGA responds to configuration reads and writes when IDSEL 20 is driven. The CPLD is not configured. Table 1 lists the values of the PCI configuration space. In order to set up the DMA engine as a PCI master, bit 2 for address 0x04 must be set to 1. Bit 1 must be set to 1 to enable writes to the Translation Buffer. Simply writing 7 to address 0x04 will accomplish both of those. At start up the device will be configured with the enable bits set. To turn off the DMA engine, a configuration write can turn off bit 2 for address 0x04. Equivalent beta functionality will require coding of a control register in its firmware.

Table 1

31	15 0	Address (hex)	Description	
0x0CDF	0x0DEA	00h	Device ID	Vendor ID
0000 0100 0000 0000	0000 0000 1000 0XXX	04h	Status	Command
0x0601	0x3748	08h	Class Code	Revision ID
0xXXXX	0x0000	10h	Base Address Register	
0x0000	0x01XX	3Ch ??		Interrupt
0x0000	0x0000	All others 14h-38h	Undefined Addresses	

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. The main memory address used must map to a part of the main memory's DMA address space. The location and size of the window is set by the FLASHROM firmware by writing to the appropriate register in the 21172-CA chip. For the EB164 there is a 1 MB DMA window starting at PCI address 0x100000. The location and size of the window are defined in the eb164.h file and the executed code is in eb164_io.c. DØ defines a new DMA window beginning at PCI memory address 0x40000000 mapping to all physical memory in its L2ALPHA device driver code.

3.2 Description of DMA operation on the Alpha boards

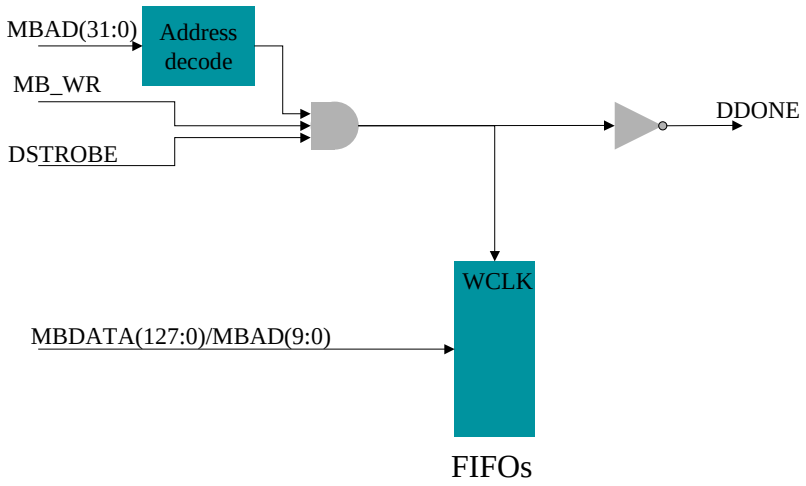
The MBT boards send their data to the processor boards over the Magic Bus by addressing one of the DMA channels in the 32-bit Magic Bus Address space. The DMA addresses are reserved to be the lowest 1024 addresses in MBUS address space. These correspond to MBUS Address bits 31:10 being set to zero.

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 Alpha board responds with DDONE*, ending that MBUS transaction[5] (Figure 1). The Alpha 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

[5] See the DØ MBus document for timing diagrams, http://galileo.phys.virginia.edu/~rjh2j/l2beta#specs.

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 1: Input logic for broadcast data



The CPLD controls the Read Clock for the data and address FIFOs. As soon as the FIFO EMPTY flag goes false, the CPLD will start sending data to main memory. The CPLD initially arbitrates to gain control of the PCI bus and starts sending data as soon as it can. The 10 bits of the DMA channel are used to address the 4 SRAM chips of the Translation Buffer (TB). The Translation Buffer asserts the corresponding 32-bit PCI address on the PCI bus. On the next PCI clock, the lower 64 MBUS data bits are asserted on the PCI bus, with the upper 64 bits sent on the following PCI clock. The DMA engine always performs a 64-bit data transfer, and always sends at least two consecutive data words following one address.

While sending the data, 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.

The Alpha's 21172-CA chip (which manages PCI access to main memory and the CPU) has a buffer that holds eight 64-bit words. When the buffer fills, the 21172 asserts a STOP signal on the PCI bus. The DMA engine responds to the signal by ending the burst transaction and releasing the PCI bus. The DMA engine then re-arbitrates for the PCI bus to continue the transaction. The DMA transaction also halts if the FIFO becomes empty.

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. 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 βetas, however we should take care to preserve compatibility with the Alpha processor online code.

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 PCI configuration space, 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. The βeta firmware must also contain a mechanism for disabling DMA.

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

There is an opportunity for a number of enhancements in the beta processor for MBus arbitration. For example:

- Inputs to the FIFOs could be software enabled/disabled
- Reporting of DDONE could be software enabled/disabled. In this case, only an administrator card could be configured to return DDONE for the crate of processors.

3. 3 Summary of DMA issues for L2βeta

- Discussion of the DMA firmware model in the L2beta TDR 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. Depending on the speed of DMA transfers through the PLX chip, the performance penalty for this may 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.

4 Magic Bus Programmed I/O

The Magic Bus Programmed I/O chip (MAGICFPGA) 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 supports up to 64 bit PCI data transfers but only 32 bit addressing, and accesses all 128 bits of MBUS data. This device is implemented in a Xilinx XC4036EX-2-PG411C.

Configuration

The MAGICFPGA is configured by writing to the appropriate configuration registers according to the PCI specification. The MAGICFPGA responds to configuration reads and writes when IDSEL 21 is driven. The MAGICFPGA asks for 3 separate 16-bit (64K) memory address spaces. 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 2 lists the values of the PCI configuration space. Bits indicated by "X" must be written to configure the device. The last 3 bits of address 04h should be written as 110 to enable memory reads/writes to the device as well as PCI bus mastership.

In principle, the PCI memory addresses should be set by the plug-n-play PCI protocol. In that case user software would simply read these configuration registers to know what PCI addresses to use for that device. However, the configuration registers can be changed at any time by user software. In that case the software should be careful to not configure the device with a PCI address conflicting with another device on the board.

Table 2

31	15 0	Address (hex)	Description	
0x2CDF	0x2DEA	00h	Device ID	Vendor ID
0000 0100 0000 0000	0000 0000 1000 0XXX	04h	Status	Command
0x0701	0x3748	08h	Class Code	Revision ID
0xXXXX	0x0000	10h	Base Address Registers	
0xXXXX (Window A)	0x0000	14h		
0xXXXX (Window B)	0x0000	18h		
0x0000	0x01XX	3Ch		Interrupt
0x0000	0x0000	All others	Undefined Addresses	

The base memory address at configuration address 10h defines the base memory address for the Magic Bus configuration or "setup" space. These setup registers are not in PCI configuration space, but are accessed through a normal PCI memory transaction. These registers hold the mapping from PCI to MBUS address space and vice versa. Table 3

describes the memory location of each of the registers. The address is the address given in the table, plus the base memory address as defined in the PCI configuration register 10h. These registers must be configured by user code for proper function of the MAGICFPGA with the Magic Bus.

Table 3

Register Description	Address (hex)
PCI Translation Base	00h
MBUS Upper Memory Address	04h
MBUS Lower Memory Address	08h
MBUS Translation Base	0Ch
MBUS Mask (not used)	10h

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

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 2** illustrates how a PCI address is converted to a MBUS address.

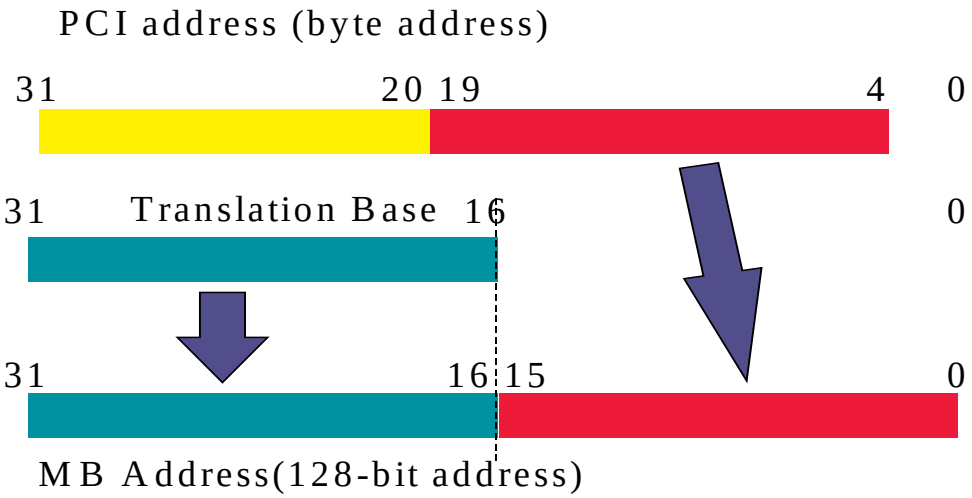


Figure 2: PCI address to MBus address conversion.

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.

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, it will assert STOP on the PCI bus. This will signal the 21172 (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 A should be mapped in the PC164's PCI sparse memory space which is defined to be in the physical address range 80.0000.0000 to 85.7FFF.FFFF. See the PC 164 TRM for details of the address space.

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 should be mapped to the dense memory space that is defined to be in the physical address range 86.0000.0000 to 86.FFFF.FFFF. The actual implementation of this in the Alpha firmware is easily described. 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 Alpha FPGA can in theory support a burst read of up to 128 bits using either 32 or 64 bit transfers. In practice, the initial PIO firmware supports only 32-bit non-burst mode transactions, so PCI four writes/reads are necessary to transfer a single 128-bit word.

Magic Bus Transactions

A transaction 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 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 transaction is finished.**

The MBUS Upper Address and MBUS Lower Address define the window in MBUS address space for which the Alpha 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 3 illustrates how a Magic Bus address is converted to a PCI address.

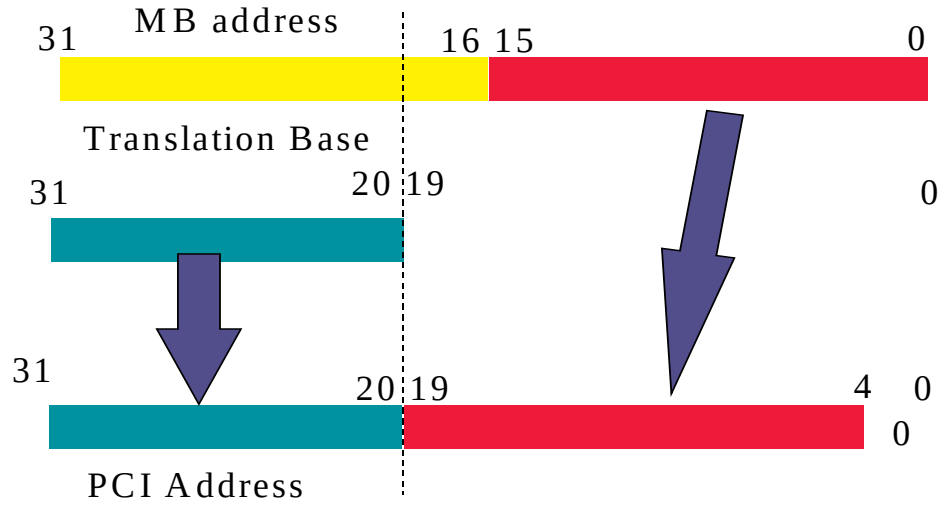


Figure 3: MBus address to PCI address conversion

MBus to PCI mapping a concrete example:

Assume the following configuration for the MBUS PIO registers:
CIA Memory base: 40000000 (This setup is specific to the Alpha PCI controller. Its function is to map PCI addresses to physical memory addresses. In our case any access to PCI addresses between 0x40000000 and 0x40000000+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: 0x41000000 (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 0x41000000. A PIO transaction at MBus address 100001 generates a PCI cycle with address 41000010.

As stated above, ALPHA firmware PIO transactions are **fully coupled** transactions. 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 Target's PCI bus and transfers data (4x32 bits or 2x64bits)[6]
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[7]
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.

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 βetas by use of control lines on the local bus of the PLX9656.

TSI Interface

The TSI interface is implemented in a Xilinx XC4010e-2PQ208c FPGA (TSIFPGA). This chip only functions as a PCI slave and is used to receive and send information to the

[6] A single 32-bit write may trigger an MBUS cycle when writing to Window A. Otherwise the MBUS transaction is triggered when the upper 32 bits of a 128-bit aligned MBUS word are filled (Window B).
See the FPGA document for more information.
[7] The necessary PCI bus STOP/RETRY arbitration should be handled automatically with proper setup of the PLX interface in the L2βeta.

rest of the trigger system. This includes the CDF/DØ trigger signals from the P2 backplane, as well as direct communication with the Global Level 2 board through a front panel connector. It is also used for monitoring the status lines of the Magic Bus.

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. With the interrupt PAL code, this corresponds to bit 2 of ISA address 805. (A second interrupt line sets the same bit in ISA space.) See page 4-15 of the PC164 Technical Reference Manual for details.

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 4.

Table 4

I/O	Alpha TSI Connection ID	Alpha J2#	Administrator Function	Worker Function
in	Ext_Test_Interrupt	A24	SCL_Initialize_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		n/a	Worker_3 Interrupt	
out		n/a	Worker_4 Interrupt	
out		n/a	Worker_5 Interrupt	

Several of the J2 pins unused by D0 should remain connected on the betas to allow for future flexibility in custom I/O definitions. Recommendations are shown in Table 6.

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.

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). The TSI may also be software enabled to generate an interrupt under either of two conditions:

1. New event Interrupt: (MOD_DONE MASK is satisfied) .AND. (LOCAL FIFO EMPTY .OR. AP FIFO EMPTY) .AND. (Interrupt 1 enabled)
2. SCL Initialize: (SCL Initialize line on J2 pulled high) .AND. (Interrupt 2 enabled)

Configuration Registers

The TSIFPGA is configured by writing to the appropriate configuration registers according to the PCI specification. The TSIFPGA responds to configuration reads and writes when IDSEL 23 is driven. The TSIFPGA asks for a 16-bit (64 K) memory address space. Table 5 lists the values of the PCI configuration space. Bits indicated by "X" must be written to configure the device. The last 3 bits of address 04h should be written as 010 to enable memory reads/writes to the device. Writing to the upper 16 bits of 10h specifies the base memory address. This address should be set by the plug-n-play PCI protocol.

Table 5: TSI Configuration registers

31	15 0	Address (hex)	Description	
0x2CDF	0x2DEA	00h	Device ID	Vendor ID
0000 0100 0000 0000	0000 0000 1000 0XXX	04h	Status	Command
0x0701	0x3748	08h	Class Code	Revision ID
0xXXXX	0x0000	10h	Base Address Register	
0x0000	0x01XX	3Ch		Interrupt
0x0000	0x0000	All others 14h-38h	Undefined Addresses	

TSI Memory Registers

The memory location of the TSIFPGA registers and the signals accessed by each bit are shown in Table 6. Each register bit is listed as an input or an output. Input bits monitor that status 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 Figure 4).

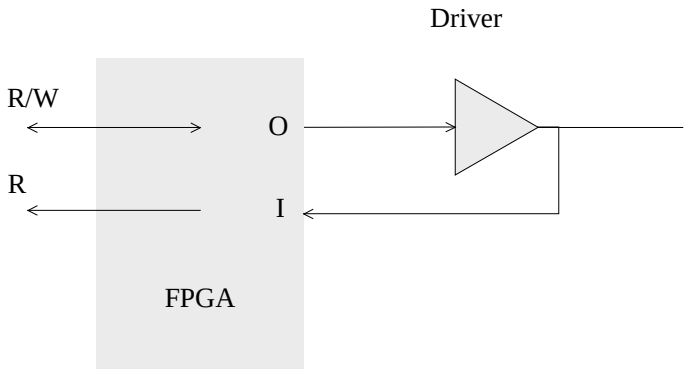


Figure 4: Driving and monitoring of output lines. Two FPGA registers bits are used in these cases, one to drive the bus and one to monitor the bus. In some cases the output driver may only be enabled if a CRATE_MASTER_BIT is set on the board. See alpha schematics (sheet 40).

Table 6: TSI Memory registers. I/O levels are shown as follows: TTL: standard TTL, TTLO64: Open collector, sinking 64mA, TTL64: TTL Driver, sinking 64mA
(functions not present in the L2Alphas are listed in *italics*)
CDF specific registers are deleted from this table

Address	Access	Bits	Description	Connection/Comment	I/O: Level
0ch	R		Broadcast Status Signals	Register to monitor J3 signals	
		18:0	MODE_DONE(18:0)	**J3** C41-B45	I: TTL
		19	FIFO_EF	Local fifo empty	I:
		20	MB_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*
		31:28	0x0		

Address	Access	Bits	Description	Connection/Comment	I/O: Level
10h				**These bits drive J3 signals. A read returns internal register status**	
	R/W	0	CDF_ERROR*	(Unused by DØ)	
	R/W	1	AP_FIFO_EMPTY	(CDF Output-Unused by DØ)	
	R/W	2	CRATE_MAS_DIR (Onboard control line)		
	R/W	3	TSI_DONE_OUT	DONE_OUT **J3** D45	O: TTL64
				The following 4 bits may only be driven in the MBUS if board is set to me crate master (see alpha schematics sheet 40)	
	R/W	4	BUFFER(1)	**J3** A41	O: TTL64
	R/W	5	BUFFER(0)	**J3** B41	O: TTL64
	R/W	6	START_LOAD*	**J3** C40	O: TTL64
	R/W	7	MBRESET	**J3** B38	O: TTL64
	R/W	8	(unused)	(unused)	
	R/W	9	VBD_START_REQUEST	**J2** A21	O: TTL
	R/W	15:10	(undefined)		
	R	16	VBD_DONE	**J2** C25	I: TTL
	R	17	AP_FIFO_EMPTY (unused)	(unused)	
	R	18	L2_ANSWER_READY	**J2** A23	I: TTL
	R	31:19	(unused)	(unused)	
14h		31:0	TSI_OUT(31:0)	32 Front Panel ECL lines	O: ECL
30h	R/W	18:0	MOD_DONE_Mask(18:0) (Set/read MOD_DONE mask For new event interrupt)		
	R/W	19	Select_FIFO_EMPTY 1: local fifo 0: MB_AP_FIFO_EMPTY	Interrupt based on: MODE_DONE, MODE_DONE_MASK, and FIFO_EMPTY	
	R/W	20	New_Event_Interrupt_Enable	1: Enable 0: Disable	
	R/W	21	Internal_Test_Interrupt_Enable	1: Enable 0: Disable	
	R/W	22	External_Test_Interrupt_Enable 1: Enable 0: Disable	External_Interrupt_Request is the J2 SCL_INIT signal	
	R/W	29:23	(undefined)		
	R/W	30	Primary_Int_Enable	1: enable INT_A 0: disable	
	R/W	31	Secondary_Int_Enable	1:enable INT_B 0: disable	

Address	Access	Bits	Description	Connection/Comment	I/O: Level
34h	R	0	Valid_Internal_Test_Request	Internal_Test_Int_Request AND Internal_Test_Int_Enable	
	R	1	Valid_External_Test_Request	External_Test_Int_Request AND External_Test_Int_Enable	
	R	2	Valid_New_Event_Request	All selected MOD_DONE lines high, AND selected FIFO_EMPTY flag high, AND New_Event_Int_Enble	
		7:3	(undefined)		
	R	8	External_Test_Interrupt_Req	**J2** A24	I: TTL
		29:9	(undefined)		
	R	30	INT_A	1: Asserted 0: not Asserted	
	R	31	INT_B	""	
3C	R/W	0	Internal_Test_Int_Req 1: Assert 0: Deassert	Internal_Test_Int_Enable AND Primary_Interrupt_Enable must also be set to assert INT_A	
	R/W	1	J2_Test_Output(0)	**J2** C21 (set by crate master)	O: TTL
	R/W	2	J2_Test_Output(1)	**J2** C22 (set by crate master)	O: TTL
	R/W	*3*	*J2_Test_Output(2)*	***J2** C32 ""*	*O: TTL*
	R/W	*4*	*J2_Test_Output(3)*	***J2** C24 ""*	*O: TTL*
		31:5	(undefined)		
40			*Additional J2 Output Lines*		
	R/W	*0:7*	*Reserved outputs*	***J2 A6-7, A9-10, A12-15***	*O: TTL*
44			*Additional J2 Input Lines*		
	R	*0:7*	*Reserved inputs*	***J2 C7-11, A22, A25, A29***	*I: TTL*
48	R	4:0	GA(4:0) Inverted values of GA# lines	**J1** D10,11,13,15,17	
		5	GAP Invert GAP# line	**J1** D9	
		31:6	(undefined)		

PCI Interrupts

The L2βeta's will implement two distinct PCI devices on the CPCI expansion bus, UII and PLX. In keeping with the CPCI recommendations for PCI interrupt binding, these devices should be routed to interrupt A and B respectively. The UII interrupt line should pass through the Xilinx FPGA on the beta – this will allow future compatibility with Message Signaled Interrupts under the PCI 2.2 standard.

Resets

See section on the Universe chip.

PCI Devices

Table 6. PCI device addresses for Concurrent Tech. CPCI board.

PCI Device	IDSEL line
CPCI Expansion	16 (factory set on primary PCI bus)
UII	16 (subordinate bus)
PLX9656	17 (subordinate bus)