1.0 Introduction

This document describes the "High performance Peripheral Controller", HPC3. This is the third generation of HPC chip. The first generation is used in Magnum and Hollywood big endian machines. The second generation is a little and big endian version of HPC1. The third generation has support for different peripherals, connects to the GIO64 bus, and does not support the DSP chip which was used with HPC1. This chip interfaces the standard peripherals to the GIO64 bus. The GIO64 bus connects the CPU and memory system to the I/O devices in the machine. HPC3 consists of a full 64 bit GIO64 interface, a few dedicated ports for specific higher-speed peripherals, and support of an external bus for generic lower-speed peripherals.

1.1 HPC3 Features

HPC3 supports a large number of different peripherals. There are six major functional blocks in the design: a 64 bit GIO64 interface, an ethernet port, two SCSI ports, the serial EEPROM interface, and the PBUS controller. The dedicated ports support the following devices:

Ethernet, Seeq 8003 and 8020 chip set
SCSI, Western Digital WD33C92A ’93A ’93B 95
SCSI, Fujitsu 86603
Serial EEPROM, National NMC93CS56

The PBUS is a general purpose I/O bus that has been designed to support a variety of common 8 or 16 bit devices. The PBUS controller includes support for the Boot PROM, battery-backed SRAM, 10 general purpose chip selects for programmed I/O, and 8 general purpose DMA channels. The access parameters for PIO and DMA are set with configuration registers in HPC3 to make it easy to add new devices to the PBUS. Typical PBUS devices include:

Audio, HAL2
Parallel Port
Real Time Clock, Dallas DS1286
Real Time Clock with NVRAM, Dallas DS1386, DS1397
Boot PROM, 16 bits wide, up to two 2 Mbyte PROMs (4M)
Interrupt Controller, INT2
Floppy Disk Controller, National PC8477
SCSI, Western Digital WD33C92A ’93A ’93B
UARTS
Keyboard/Mouse

HPC3 has a full 64 bit GIO64 bus interface in order to reduce the amount of time needed to provide DMA service to all of the attached peripheral devices.

1.2 HPC3 Peripheral Bandwidth

The amount of GIO64 bus bandwidth that HPC3 uses depends a lot on the devices that are connected to it and how many of those devices are active. To calculate the size of the fifos for the various DMA channels it is necessary to know the maximum bandwidth of each device and how often HPC3 gets serviced on the GIO64 bus. Since it is a real time device, HPC3 can get the GIO64 bus every 30 microseconds. Therefore, it can get serviced 33,333 times a second. The maximum bandwidth and the minimum fifo size (in bytes) for each device is listed below:

Device	Mbytes/Second	Fifo Required	Fifo on Part
Ethernet In	1.25	38	16
Ethernet Out	1.25	38	16
SCSI	20	600	12
Parallel	1	30	0
Floppy	0.125	4	16
Audio In	0.192	6	4
Audio Out	0.192	6	4
PBUS Maximum	8	240	-

The audio rates above represent a 32 bit sample (16 bit stereo) at 48 KHz. Refer to the HAL2 specification for actual audio bandwidths. The maximum total I/O bandwidth of HPC3 is 30.5 Mbytes/second which occurs when the following devices are connected:

2 Channels Fast SCSI	20
Ethernet	2.5
PBUS	8

The total amount of *fifo ram* (in bytes) in HPC3 is:

2 SCSI Channels	2 @ 384
Ethernet transmitter	1 @ 160
Ethernet receiver	1 @ 128
PBUS	1 @ 384

The Ethernet transmitter and receiver fifo rams are split into two ping pong buffers. So, the effective size is 80 and 64 bytes respectively.

When HPC3 gets the GIO64 bus it services ethernet first, then the PBUS DMA channels, and finally the two SCSI ports. This order is used so that the ethernet and audio bus request to service times will be reduced. When a fifo is to be serviced from the GIO64 bus, any device that is using the fifo may continue to do so because the internal rams which are used to implement the fifos are multi-ported.

1.3 HPC3 Chip Technology

HPC3 is packaged in a 299 pin Ceramic Pin Grid Array, with provisions that the same die can be packaged in a 304 pin Metal Quad Flat Pack when that package is qualified. The random logic gate count is approximately 58,000 gates. Because of the large amount of on-chip memory (>1600 bytes, there are 3 rams in addition to the *fifo* rams) the chip is designed using LSI Logic's 1 micron embedded array technology. It uses the largest die at 15 mm square. Phase-Locked Loop and LSI's Clock Compiler are used to minimize system and on-chip clock skew respectively. ATPG and a modified boundary scan approach are used to facilitate manufacturing test.

1.4 Bit and Byte Numbering Conventions

This chip operates in both big and little endian modes. Which mode it operates in depends upon the operation being performed. For DMA, the endian mode is determined by a DMA channel configuration register. The endian mode of a DMA operation affects the GIO64 DMA data transfer and the packing or unpacking of data between the device and the fifo in HPC3. Different DMA channels can function using different endian modes. For example, one SCSI channel can be running in little endian mode, while the other is in big endian mode. DMA operations are descriptor based, and therefore HPC3 needs to know how to read the DMA descriptors from main memory. The endian mode used for **all** DMA descriptor fetches will be set by a **global** configuration register in HPC3. The endian mode of CPU reads and writes is determined by the GIO64 endian mode bit that is sent during the GIO64 byte count cycle, so processor loads and stores can be of either endian.

It is important to know the difference between big and little endian mode. Big endian means that byte 0 is bits (63:56), byte 1 is bits (55:48), byte 2 is bits (47:40), byte 3 is bits (39:32), byte 4 is bits (31:24), byte 5 is bits (23:16), byte 6 is bits (15:8), and byte 7 is bits (7:0). Little endian is just the opposite, so for a 64 bit little endian transfer, byte 0 is bits (7:0), byte 1 is bits (15:8), byte 2 is bits (23:16), byte 3 is bits (31:24), byte 4 is bits (39:32), byte 5 is bits (47:40), byte 6 is bits(55:48), and byte 7 is bits (63:56).

The bit numbering scheme is always little endian, so that bit 0 is always the least significant bit and bit 63 is always the most significant bit. Bit fields within a register should **never** be ordered based on endian mode.

1.5 Signal Naming Conventions

Signal names that end with a trailing underscore n, "_n", denote signals that are active low. All other signals are active high.

1.6 Definitions

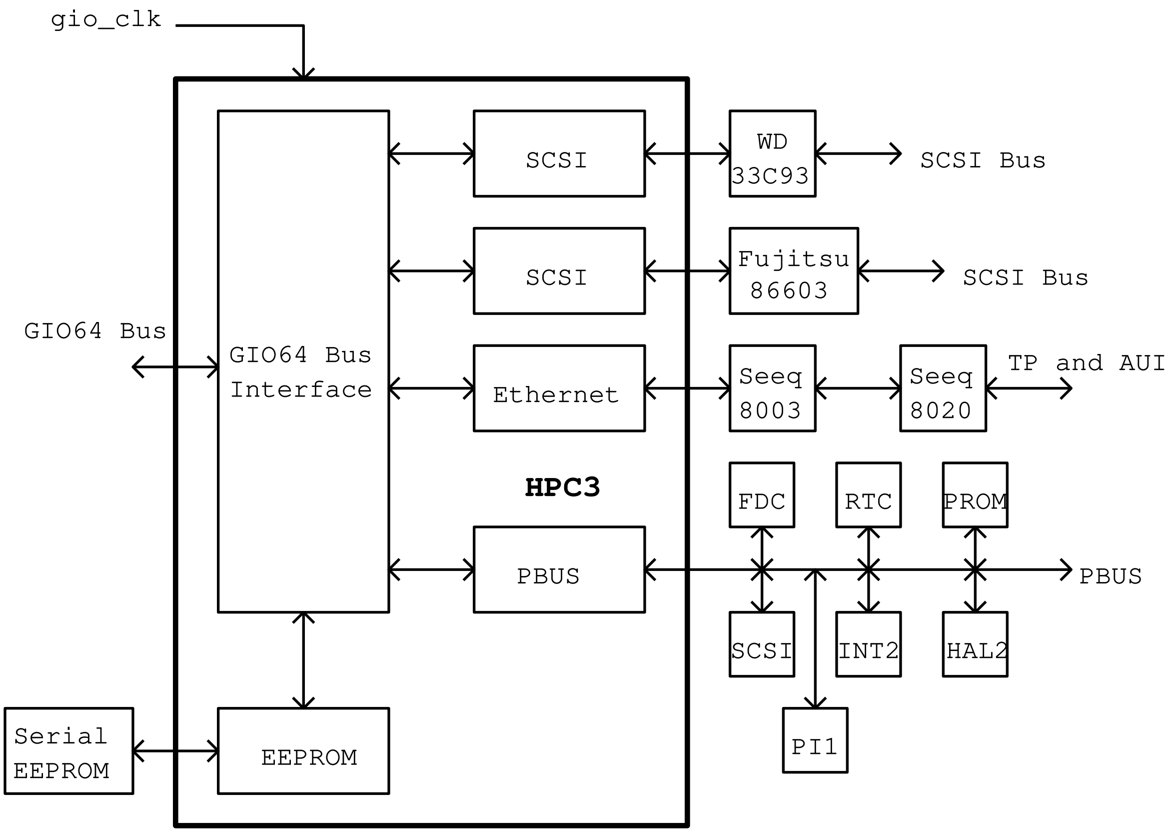
DMA Read, Receive a DMA transfer from device to main memory.

DMA Write, Transmit a DMA transfer from main memory to device.

Page the smaller of the physical page size and the DRAM page size. The page size is either 4k or 8k bytes.

2.0 HPC3 Functional Description

There are six major blocks in the HPC3 design: the GIO64 bus interface, two SCSI ports, an ethernet port, the PBUS controller, and the serial EEPROM interface. The GIO64 bus interface connects the HPC3 chip to the CPU and main memory. The two SCSI ports can be connected to either a WD33C93 controller (8 bit interface) or to a Fujitsu 86603 controller (8/16 bit interface). The ethernet port supports the Seeq 8003 controller chip. The PBUS is designed to support a large number of different devices. The serial EEPROM is a ROM that contains the boot monitor environment information and chassis serial number. A block diagram of the HPC3 with some attached peripherals is shown below. HPC3 receives one single clock input. The ***gio_clk*** goes through a PLL to minimize system clock skew.



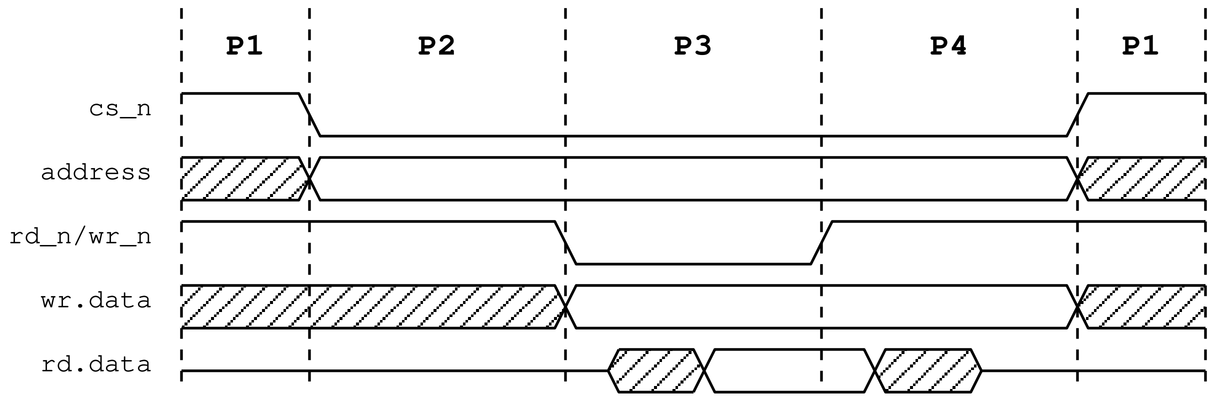
HPC3 Block Diagram

2.1 HPC3 Programmed I/O

The HPC3 is a GIO64 bus slave during programmed I/O operations. There are basically three types of PIO operations - register access, dma fifo ram access, and PROM access. The HPC3 address map is shown in section 3. An HPC3 ***register*** is simply any addressable ***word*** in HPC3 address space, except those in the Boot PROM address space or fifo rams. This includes all *internal* HPC3 registers, and all *external* peripheral registers addressable through the HPC3. The external devices are really 8 or 16 bits wide. In this case, the actual data transfer applies to bits (7:0) or (15:0) of the *word* register. Byte/halfword oriented code will be endian-sensitive! Word oriented code will not be endian-sensitive! The on-chip rams used to implement the dma fifos are in the HPC3 address space as indicated in section 3. They are accessed as doublewords (this is a testability feature). The PROM consists of addressable halfwords, words and doublewords in the Boot PROM address space.

All registers (internal and external) are 32 bit words with word aligned addresses from the GIO64 bus perspective. The two least significant register address bits are ignored. For all register PIO, a single 32 bit word will be transferred. All unused bits in a register will read back as 0, except for PBUS external registers. PBUS external registers are replicated to fill the word, e.g., for 8 bit devices the word returned will be four copies of the same 8 bits. GIO64 block transfers are not supported for HPC3 PIO. If the GIO64 byte count is less than 4, the transfer will still be 4 bytes. If the byte count is greater than 4, HPC3 will transfer one word, drive ***gio_slvdly*** active, and then tristate ***gio_slvdly***. In this case, after the first word is transferred, the GIO64 bus will be waiting for ***gio_slvdly*** to be deasserted, and eventually it will time out.

Access of peripheral registers, or the PROM, requires that the HPC3 perform programmed I/O externally with the addressed device. The diagram below shows the general nature of this device PIO. The states indicated (P1 to P4) correspond to certain device specific timing parameters, and sometimes the number of ***gio_clk*** cycles for a state can be controlled using an internal configuration register. Note that the SCSI and Ethernet PIO waveforms deviate slightly from the specific timing parameters shown here. Please refer to the SCSI and Ethernet sections for more details.



Device PIO Waveforms

2.2 HPC3 Direct Memory Access

The DMA model described in this section applies generally to all HPC3 DMA channels. Any variations are pointed out in the sections which follow. Like the earlier generations of HPC, HPC3 DMA operations are descriptor based. A DMA descriptor is a linked list structure in main memory that describes a DMA operation. Before the DMA operation begins, all necessary external device registers and internal DMA channel registers (including a pointer to the first descriptor in a chain) are configured. Then the start DMA bit for the DMA channel is set. At this point, the DMA channel is said to be active. HPC3 will fetch the first descriptor and start the DMA operation. Each descriptor consists of 3 consecutive 32 bit words. All descriptors must be quadrupleword aligned in main memory, and must not cross a page boundary. The figure below shows the fields of a descriptor and the organization of a descriptor in main memory. The buffer pointer (BP) word is always at the least significant word address, followed by the byte count (BC), and then the descriptor pointer (DP) at the most significant word address.

<table>
<tr><td>0x0</td><td colspan="8">Memory Buffer PHYSICAL Address(31:0)</td><td>BP</td></tr>
<tr><td>0x4</td><td>EOX</td><td>EOXP</td><td>XIE</td><td>RES(28:24)</td><td>IPG(23:16)</td><td>TXD</td><td>RES(14)</td><td>Byte Count(13:0)</td><td>BC</td></tr>
<tr><td>0x8</td><td colspan="8">Next Descriptor PHYSICAL Address(31:0)</td><td>DP</td></tr>
</table>

DMA Descriptor Format

EOX BC(31) indicates that this is the last descriptor in the chain. After completely processing this descriptor, the DMA channel becomes inactive. The DP field is not used for this descriptor.

EOXP BC(30) tells the ethernet controller that this descriptor represents the end of a packet. This only applies to ethernet transfers. EOXP should always be set for ethernet receiver descriptors (there is exactly one DMA descriptor per ethernet receive packet).

XIE BC(29) tells the HPC3 to generate an interrupt after processing this descriptor. For the ethernet controller, XIE is ignored if EOXP is not set.

RES BC(28:24,14) bits are reserved and are not used.

IPG BC(23:16) is a byte used only for the Ethernet transmitter. This byte is written to SEEQ 8003 extended register x02 after either HPC3 has moved the first 80 bytes of a packet (one fifo buffer size) to the SEEQ 8003 or HPC3 has moved one complete packet (shorter than 80 bytes). This feature is enabled by a bit in the Ethernet DMA configuration register.

TXD BC(15) is a bit used only for the Ethernet transmitter. This bit is written to a ’1’ in the 1st descriptor for a packet after the packet has been completely transmitted (through the SEEQ 8003 controller). In all other cases, this bit should be a zero.

Byte Count BC(13:0) indicates the number of bytes to be transferred. This is the size (in bytes) of the main memory data buffer. In the case of an Ethernet receive DMA, this field is written back after this descriptor has been completely processed with a value which reflects the actual number of bytes received (see Ethernet section for more details).

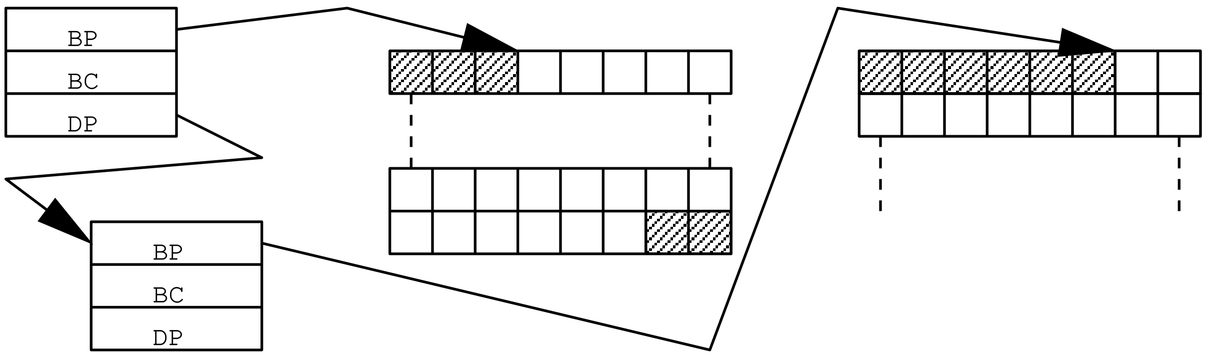
Memory Buffer PHYSICAL Address points to the buffer in main memory to transfer DMA data to or from. This buffer has a maximum size of one page, and cannot cross a page boundary. Ethernet receive buffers must always start on a doubleword boundary. This must be a physical address!

Next Descriptor PHYSICAL Address points to the next DMA descriptor in main memory if the EOX flag is not set. This is the *link* to the next descriptor to process. This must be a physical address!

For PIO, HPC3 is the GIO64 bus slave. For DMA, HPC3 will become the GIO64 bus master for two distinct operations - DMA descriptor fetch and DMA data transfer.

After the HPC3 fetches and interprets the DMA descriptor, the DMA transfer begins. The transfer between device and memory is done in two steps, with an intermediate stop in HPC3 along the way. During a DMA read operation, HPC3 reads data from the device and stores it in an internal fifo. It then initiates a GIO64 cycle which writes the data from the fifo to the main memory buffer. During a DMA write operation, HPC3 initiates a GIO64 cycle which reads data from main memory and stores it in an internal fifo. It then writes the data from the fifo to the device. HPC3 never does a DMA transfer with the DMA Count Direction Down bit asserted (see GIO64 bus specification for more details).

The diagram below shows the DMA descriptor and main memory buffer structure. For DMA write buffers, there are no alignment constraints. HPC3 will pack together bytes from one buffer to the next in a seamless fashion while storing them in the fifo. For DMA read buffers however, the starting byte address of one buffer must be aligned with the byte after the end of the previous buffer. The buffers do not have to be contiguous, but they must be aligned with respect to one another - hence they *appear* contiguous. This is required because HPC3 reads data from the device and packs it into the fifo without knowing where the seams are from one buffer to the next, or even the main memory buffer start address from the next descriptor.



DMA Descriptor and Main Memory Buffer Structure

There are several configuration bits common to all DMA channels. The start DMA bit mentioned above is actually an indication that the channel is active. Clearing this bit will cause the channel to become inactive, effectively aborting the current operation. When a DMA operation completes normally, HPC3 will clear this bit so that the channel becomes inactive.

Each channel has a configuration bit which indicates the endianess of the current transfer, big or little. This bit can be changed between DMA operations, but should not be changed while the channel is active. There is one **global** configuration bit which establishes the endianess of **all** DMA descriptors.

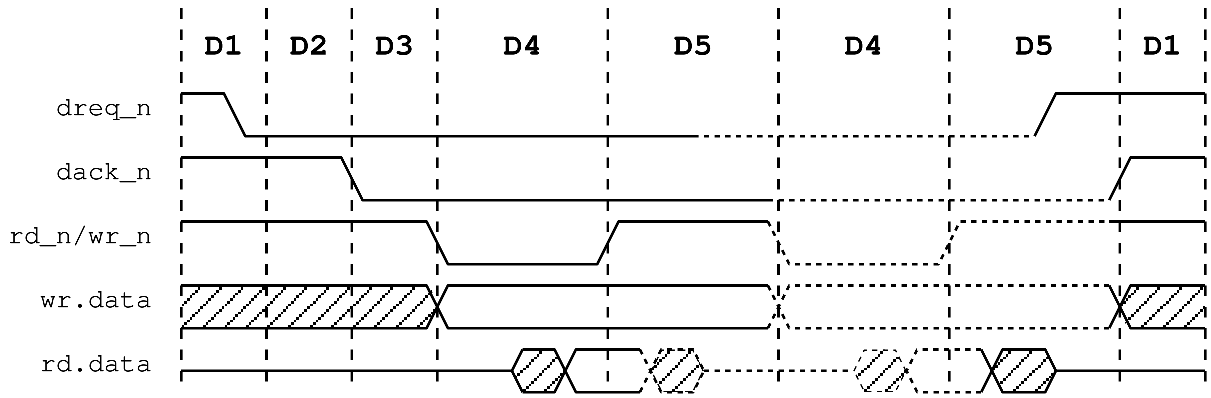
Most channels have a configuration bit which indicates the direction of the current transfer, receive or transmit. Again, this bit should not be changed while the channel is active. The two Ethernet channels do not contain direction bits as the direction is implicit.

Each channel also has a flush bit which will stop receive DMA between HPC3 and the external device. Any remaining bytes in the fifo will be flushed to main memory and then the DMA channel will deactivate itself. All remaining data in the fifo will be flushed unless the end of the descriptor chain is reached first. If there is more data in the fifo than the current main memory buffer will hold, and the EOX bit is not set, the HPC3 will continue to process the linked list of descriptors. The flush will continue until either the descriptor chain or the data in the fifo is exhausted. Any interrupts called for by the descriptors will still be generated.

When one of the DMA channels in HPC3 finishes processing a descriptor with the XIE bit set, an interrupt will be generated. All DMA channels, except for ethernet share one interrupt pin on HPC3. Some channels will not need to generate this interrupt, since the controller they are connected to can generate its own interrupts. The Western Digital SCSI chip, for example, will generate interrupts on its own behalf, so the XIE interrupt feature may not be needed. The interrupt status of each DMA channel can be read from two registers in HPC3.

The exact time at which the interrupt is generated depends on whether the operation is a read or a write, and whether the EOX bit is set. For DMA read operations with XIE set, the interrupt will be generated when the last byte of the main memory buffer has been written. For DMA write operations with XIE and EOX both set, the interrupt will be generated when the last byte is transferred to the device. For write operations with XIE set, but EOX not set, the interrupt will be generated when the last byte of the main memory buffer has been read and put into the HPC3 fifo. The interrupt is generated at this time because the fifos are maintained in a seamless fashion; consequently, the correspondence of fifo_byte-to-main_memory_buffer is not known.

As already indicated, the DMA data transfer takes place in two steps - GIO64 DMA transfers and external device DMA transfers. For information on the GIO64 data transfers see section 2.3 GIO64 Bus Interface, and the GIO64 Bus Specification. The diagram below shows the general nature of the external device DMA. The states indicated (D1 to D5) correspond to certain device specific timing parameters, and sometimes the number of ***gio_clk*** cycles for a state can be controlled using an internal configuration register. Note that the SCSI and Ethernet DMA waveforms deviate slightly from the timing parameters shown here. Please refer to the SCSI and Ethernet sections for more details.



Device DMA Waveforms

2.3 GIO64 Bus Interface

HPC3 is a 64 bit device and will be able to run the GIO64 bus at 33 MHz. It will support the nonpipelined GIO64 bus protocol with parity checking (bus destination), parity generation (bus source), big/little endian addressing, and block transfers (DMA accesses only). HPC3 is a bus slave for all programmed I/O transfers and a bus master for all DMA transfers.

Although HPC3 does not support bus preemption, HPC3 can be kicked off of the GIO64 bus through use of the gio_timeout_n pin.

2.3.1 Programmed I/O

HPC3 is a GIO64 bus slave during programmed I/O cycles. There are basically three types of PIO operations - word register access, dma fifo ram access, and PROM access. These three types of PIO were described in section 2.1.

All register accesses are word accesses. All dma fifo ram accesses are doubleword accesses. PROM accesses are halfword, word, or doubleword accesses. In all of these cases, there is exactly one data transfer. After this single data transfer the HPC3 will assume the access is complete and get off the bus. If the GIO64 bus master is not done with the transfer because it used a bad byte count, the GIO64 bus will timeout. Byte and halfword oriented code will be endian sensitive! Word oriented code will not be endian sensitive! This is because each register access will be implemented as a word access regardless of the byte count.

***** BUG *****
There is a problem with the PIO read back of all DMA descriptors. Before reading any of the DMA descriptor ports, software has to flush the single stage PIO write buffer in HPC3. This can be done by doing a PIO read from any register immediately before the DMA descriptor read. Note that software must guarantee that these PIO reads will occur back to back. Failure to do so will result in the wrong data being returned during the PIO read.

2.3.1.1 Single Stage Write Queue

There is a one stage write queue to allow release of the GIO64 bus immediately. During reads, HPC3 must stall the GIO64 bus until the proper register can be accessed from the device. If it is a HPC3 on chip port, the stall should be minimal.

PIO writes to external device registers may take very long. HPC3 does not wait until these writes are complete before responding to a PIO read/write for an on-chip port. HPC3 will stall until the write is complete before responding to a PIO read for a external register from the same device (only from the SAME device). For any one device, all PIO writes and PIO reads will occur in the order in which HPC3 received the requests. For all on-chip ports, all PIO writes and PIO reads will occur in the order in which HPC3 received the requests. However, the order of requests may not be maintained between external devices and on-chip ports.

2.3.2 DMA

HPC3 is a GIO64 bus master during DMA cycles. When transferring data from memory to a device, consecutive blocks of data are "packed" together into the device fifo (on the HPC3). This allows for a seamless transfer of data from the device fifo (on the HPC3) to the device itself.

There is a single common GIO64 interface for all of the HPC3 devices. One arbiter constantly polls all of HPC3's DMA channels to see if any need service. Upon detecting that a device needs the bus, ***gio_breq_n*** is asserted on the GIO64 bus. When the bus request is granted (***gio_bgnt_n*** is returned), the cycle is given to the device with the highest priority needing service. Priority is fixed. When this device is finished, control is passed on to the device with the next highest priority. This repeats until each HPC3 DMA channel is serviced or HPC3 has exceeded the maximum time it can hold the bus (see section below). HPC3 then cedes control of the GIO64 bus.

Writing a programmed I/O register, which affects the DMA channel, while the channel is active is not recommended. Software should wait until the channel is inactive (***ch_active*** is inactive) before programming any of these registers. If one of these registers is written while the DMA channel is active, there are no guarantees for the actions of the HPC3.

HPC3 will not use the DMA Count Direction Down or the CPU Subblock Ordering features of the GIO64 bus transfer protocol. These signals will always be inactive during the byte count cycle (see GIO64 bus spec. for more detail).

2.3.3 Timer on GIO64 bus hold

HPC3 is a real time device in the system. HPC3 will never be preempted off the GIO64 bus through GIO64 protocol (gio_preempt_n). However, there is a way to boot HPC3 off of the bus. The input ***gio_timeout_n*** (being asserted, = 0) will cause HPC3 to release the bus after it is finished with the current DMA transfer unless a real time device request is pending. If a real time device request is pending, and the **real_time** bit (***gio_misc*** register) is enabled (=1), HPC3 will continue to process the real time DMA transfers. After HPC3 is finished with all real time channel requests, it will relinquish the bus. If the **real_time** bit is not enabled, HPC3 will relinquish the bus after the current DMA transfer

2.3.4 Device Identification

When HPC3 is the GIO64 bus master, it puts a device identification out on the GIO64 bus during the byte count cycle. The device identification corresponds to which DMA channel "owns" the current GIO64 cycle. See GIO64 specification for more details.

Device ID	DMA Channel
0x0f	PBUS Channel 0
0x0e	PBUS channel 1
0x0d	PBUS channel 2
0x0c	PBUS channel 3
0x0b	PBUS channel 4
0x0a	PBUS channel 5
0x09	PBUS channel 6
0x08	PBUS channel 7
0x07	SCSI channel 0
0x06	SCSI channel 1
0x05	Ethernet Receiver
0x04	Ethernet Transmitter.

Note that when there are 2 HPC3s in the system, it may not be possible to determine which HPC3 cau

2.3.5 Parity

HPC3 generates and checks odd parity on the GIO64 bus. If HPC3 detects a parity error, it latches certain information and will interrupt the host (through the **bus_error_intr** pin). The parity error status information can be read from gio.bus_error. See the GIO64 specification for more details.

2.4 Ethernet

HPC3 will support the SEEQ 8003 ethernet controller. There are two DMA channels, one transmit and one receive, and one programmed I/O channel. The interface to the GIO64 bus is 64 bits; the interface to the SEEQ 8003 controller is 8 bits.

2.4.1 Transfer Rates

The maximum rate for the SEEQ 8003 ethernet controller is 3.3 Mbytes/s *(although the practical maximum rate for ethernet is 2.5 Mbytes/s, 1.25 Mbytes/s per channel)*. HPC3 tries to achieve this transfer rate by having both programmed I/O and DMA transfers take 11 clock cycles with a ***gio_clk*** of 33 MHz (3 Mbytes/s), or 8 clock cycles with a ***gio_Clk*** of 25 Mhx (3.1 Mbytes/s).

2.4.2 Memory structure and DMA transfers

DMA descriptors are used to control main memory transfers. There are seperate DMA descriptor linked lists for the transmit and receive channels. There is no limit on the number of descriptors to be used. Ethernet transfers data in packets. A packet may range in size from 64 bytes to 1518 bytes. When transmitting data, more than one packet may be contained within a descriptor chain and more than one descriptor buffer may make up a packet. When receiving data, more than one packet may be contained within a descriptor chain, but there can be only one descriptor buffer for each packet.

The processor starts the DMA operation by writing the **ch_active** bit for the appropriate channel. Upon detecting the **ch_active** bit is active, two things happen:

* fifo pointers are reset
* GIO64 bus request for 1st DMA descriptor

2.4.2.1 Transmit Channel

After fetching the 1st descriptor, HPC3 will fetch bytes from the 1st main memory buffer. The number of bytes in each gio64 bus transfer is either 80 bytes (size of fifo buffer) or the current byte count (starts out equal to the byte count in the DMA descriptor, and is updated at the end of each gio64 bus transfer), whichever is less. When the current byte count is zero, HPC3 will fetch the next DMA descriptor unless the EOX bit is set.

The EOX=1 flag notifies the HPC3 that this is the last descriptor to process. As soon as the HPC3 has processed this descriptor, it stops reading from main memory. When the HPC3 receives acknowledgement for all the packets it has tranmitted to the SEEQ 8003 chip, it clears the **tx_ch_active** bit. The acknowledgement comes as an interrupt from SEEQ.

During one DMA operation, many packets may be transmitted. The **eoxp**=1 flag in the DMA descriptor tells the HPC3 that this buffer is the end of a packet. When transferring the last byte of this buffer, HPC3 asserts ***enet_d(8)*** which tells the SEEQ chip that this is the end of a packet. When HPC3 receives acknowledgement for this packet from the SEEQ 8003 (the interrupt again), HPC3 sets the ***TXD*** bit in the first DMA descriptor for this packet. Note that bits 14:0 in the byte count field will be cleared when the TXD bit is written, and that this is a gio64 bus operation.

Additionally, because there are other devices contending for the network and it is not possible to determine in advance if there may be a contention problem (collision), it sometimes is necessary to stop the current transmission and restart it. HPC3 seperates collisions into two categories, early and late. An early collision is one where ***enet_txret*** is asserted before HPC3 has transferred the first 80 bytes of a packet to SEEQ. Everything else is considered a late collision.

For early collisions, HPC3 still has all of the packet data on chip (fifo buffer was sized at 80 bytes for this purpose). Upon receiving an active ***enet_txret***, HPC3 resets the device fifo pointer to the beginning of the fifo buffer. Note that although the collisions are flagged by the SEEQ interrupt going active, it is assumed that ***enet_txret*** will go active with every collision also. HPC3 then continues with the DMA transfer.

As soon as HPC3 has finished with a fifo buffer, the data held in the fifo is discarded. As a late collision is defined as receiving ***enet_txret*** after the first 80 bytes (size of fifo buffer) has been transferred to the SEEQ controller, when a late collision occurs, HPC3 does not have the start of the packet on chip. Upon registering a late collision, HPC3 resets the channel, and interrupts the host. The transmitter status register will have the late collision status bit set. One possible race condition occurs if another interrrupt occurs before the transmitter status is read.

A feature which was added to HPC3 is the ability to write the interpacket gap value (time SEEQ waits between packets) on a per packet basis. The value written is stored in the DMA descriptor (byte count field, bits 23:16). When HPC3 has moved either 80 bytes (past the late collision marker) or an entire packet, HPC3 will write the interpacket gap register. This enables us to tailor the time between packets to the packet length (or maybe other attributes of the packet). This feature is enabled by a bit in the DMA configuration register (**wr_ctrl**).

Whenever ECHRST is programmed active, the ch_active is reset..

2.4.2.2 Receive Channel

After fetching the 1st descriptor, HPC3 will start taking bytes from the SEEQ controller. The main memory buffer must always start on a doubleword boundary. HPC3 puts the 1st byte in the packet at location x02 in the fifo buffer. This aligns the data portion of the packet on a doubleword boundary (in a packet, the 1st 14 bytes are preamble, information like the source, destination, etc). When one fifo buffer is completely full or if the end of packet bit (eop) is encountered, HPC3 will request the gio64 bus.

The EOX=1 flag should never be encountered. The receive channel DMA descriptors form a ring buffer and the channel will always be "ready" to receive data. As it is impossible to know in advance the number of bytes to transfer, each descriptor’s byte count field is programmed to a value which is larger than the maximum ethernet packet size. HPC3 continues to fill it’s fifos with data from the SEEQ controller until it detects that ***enet_d(8)*** is active. ***enet_d(8)*** signifies the end of a packet. When HPC3 has transferred this last byte to main memory, it will write the (RBC - (number of bytes in the packet + 3)) into the first word of the DMA descriptor. The device driver can then poll this field in the descriptor to see whether it has been processed.

If HPC3 comes to a descriptor with the EOX=1 flag (the end of the linked list), HPC3 will clear the ***rx_ch_active*** bit in the ethernet control register upon completion of the transfer associated with that descriptor. This case should really never happen as the DMA descriptor chain should be arranged as a ring buffer.

The number of bytes allocated in each receive buffer should be a multiple of eight and should be able to accumulate a full Ethernet packet and three extra bytes. The first sixteen bytes of the receive buffer consist of two don't care bytes, destination address (6 bytes), source address (6 bytes), and byte count (2 bytes). The don't care bytes are used to pad the informative data to be on a doubleword boundary. The last byte in the receive buffer (the byte after the receive packet) is used to store the status of the received packet.

Receive discards have been seperated into two categories. An early ***rxdc*** happens before the 1st 64 bytes (based on the fifo buffer size). Everything else is considered a late ***rxdc***. Upon receiving an early ***rxdc***, HPC3 resets the device fifo pointer. If fix_rxdc is enabled, the eop status bits (eop_in_chip and rcv_eop_intr) are cleared. HPC3 then continues to receive bytes. If a late ***rxdc*** comes in, HPC3 may have already transferred bytes to main memory. In this case, HPC3 sets both eop status bits (eop_in_chip and rcv_eop_intr), which will mimic an end of packet. Late rxdc is flagged in the status byte (appended to the end of the aborted packet). The packet is written out to main memory. HPC3 then continues on.

Whenever ECHRST is programmed active, the receive channel is reset.

2.4.3 Urgency of requests

Ethernet is a real time device; software controls the transmit channel, but the receive channel is a slave to other ethernet stations. Therefore, the HPC3 must be ready to service the SEEQ 8003 upon demand. The Ethernet DMA channels will be given the highest priority, followed by the audio channels, in vying for the GIO64 bus. Also, if ***gio_timeout_n*** is asserted, but there is an Ethernet request pending and real_time = 1, HPC3 will ignore ***gio_timeout_n*** until the Ethernet channel has been serviced. If real_time = 0, then HPC3 will relinquish the GIO64 bus after the current transfer.

2.4.4 Interrupts

2.4.4.1 SEEQ 8003 register contents

The SEEQ transmit command register bits 3 down to 0 should be programmed with a '1'. The receive command register bits 5 down to 1 should be programmed with a '1'. This enables SEEQ interrupts for all frames and error conditions. HPC3 screens interrupts received from the SEEQ 8003 for the host. Upon receiving an interrupt (***enet_int_in*** = 1) from the SEEQ 8003, HPC3 will poll the SEEQ to find the condition which triggered the interrupt. Depending upon what the conditions are, HPC3 may decide to interrupt the host. The interrupt status bits from the SEEQ (transmit status register and receive status register) are copied into two HPC3 ports (Ethernet transmitter status register and Ethernet receiver status register) which the host can access.

2.4.4.2 Transmitter interrupts

There is an XIE bit in the DMA descriptor to tell the HPC3 whether to interrupt the CPU after the current tranfer is done. XIE is valid only for the last descriptor of the transmitted packet. Upon seeing this flag set, HPC3 will wait until the packet is transmitted without collisions through the SEEQ chip and then interrupt the CPU.

The only other transmit conditions which can cause an interrupt are:

* 16 failed transmission attempts (collision detected)
* Transmit underflow (should never happen)
* Late Collisions

In all cases, the ***tx_ch_active*** bit in the control register will be cleared and transmission will be stopped. HPC3 continues to process the interrupts from the SEEQ chip although the ***tx_ch_active*** bit is a ’0’. Note that the status for the condition which caused the original interrupt may be overwritten by subsequent interrupt conditions.

2.4.4.3 Receiver interrupts

The CPU will be interrupted after every active ***enet_d(8)*** (end of packet) received from the SEEQ chip (the XIE bit should be set in each DMA descriptor). HPC3 waits until the packet is sent to main memory before generating an interrupt. Hpc3 used to have a timer which would delay the interrupt. The timer was eliminated because of gate count constraints. The interrupt now occurs as soon as the last piece of data in the packet is written to main memory.

There are 3 other conditions for the receiver to interrupt the CPU. They are:

* There is a receive fifo overflow in the SEEQ 8003 chip.
* HPC3 has reached the end of the linked list of descriptors (the EOX flag is set) and has flushed the particular packet to main memory.
* HPC3 receives a packet whose number of bytes is more than the RBC. This type of error condition is flagged by the "RBO" (receive byte overflow) bit in the control register. This bit can be cleared by writing a ’1’ to it.

In all three cases, HPC3 clears the ***rx_ch_active*** bit. HPC3 continues to process the interrupts from the SEEQ chip although the RSTRTDMA bit is a ’0’. Again, note that the status for the condition which caused the original interrupt may be overwritten by subsequent interrupt conditions.

2.4.5 Implementation

Each channel, transmit and receive, has two buffers called ibuf0 and ibuf1. The ethernet main memory state machine handshakes with the GIO64 bus state machine to transfer data between the ethernet fifos and main memory. The ethernet main memory state machine controls transfers from main memory to the transmit ibuf0 and ibuf1 fifos, and transfers to main memory from the receive ibuf0 and ibuf1 fifos. The ethernet channel state machine controls the transfers between the SEEQ 8003 and the HPC3 ethernet fifos. Tranfers from the 8003 to the receive ibuf0 and ibuf1 fifos, and transfers to the

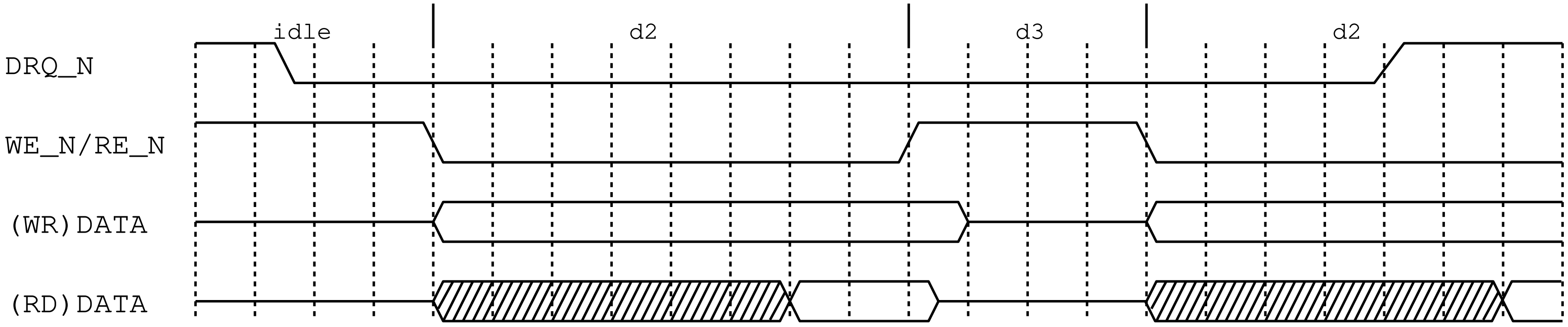
8003 from the transmit ibuf0 and ibuf1 fifos are handled by the channel state machine. The channel state machine also handshakes with the GIO64 bus to do programmed I/O with the SEEQ 8003 control registers.

2.4.5.1 Fifo Buffers

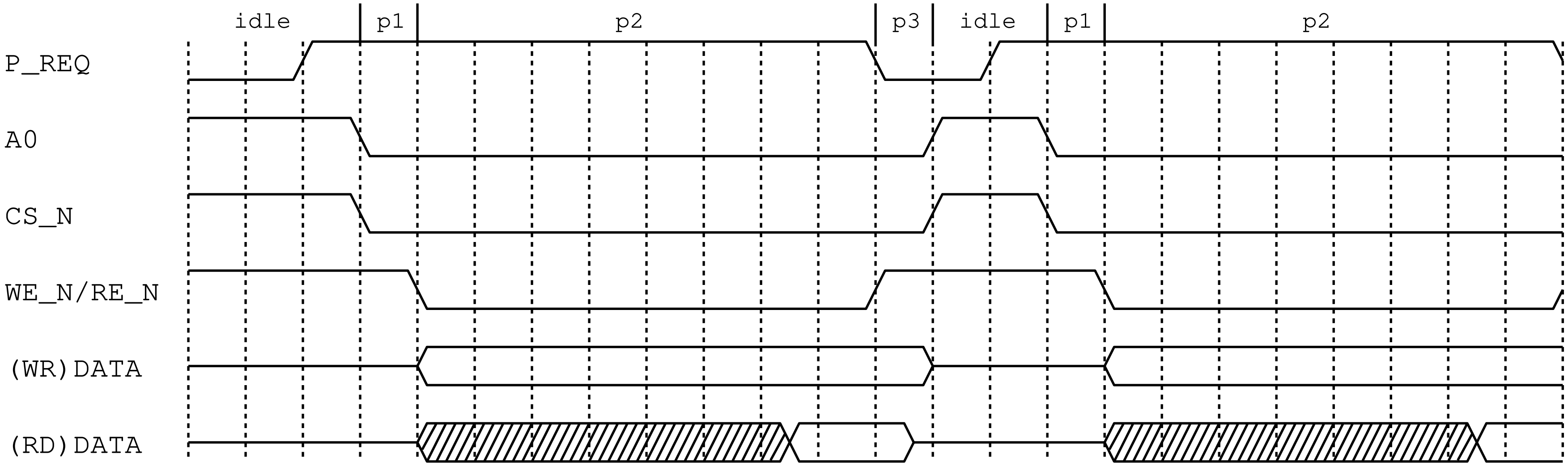
The Ethernet fifo buffers have one dedicated read and one dedicated write port. For the receiver, the channel state machine will always have control of the write port and the main memory state machine will always have control of the read port. For the transmitter, the channel state machine will always have control of the read port and the main memory state machine will always have control of the write port. It is possible to have the main memory state machine and the channel state machine both accessing data at the same time. As there are two ping-pong buffers for each channel, the main memory state machine could be working on ibuf0 and the channel state machine could be working on ibuf 1.

The high water mark (time to request GIO64 bus) is when one ping-pong buffer is completely filled (receiver) or completely empty (transmitter). The bandwidth needed for Ethernet is based on the size of one ping pong buffer. In the case of the transmitter, if we request the GIO64 bus when one buffer is empty, we need to service that buffer (fill it up) before the other ping-pong buffer is emptied. During transmission, the one byte is drained every 800 ns. HPC3 needs to be serviced every 64 us (800 ns * 80 bytes) to satisfy the transmitter channel. The ping-pong buffer for the receiver channel is slightly smaller, 64 bytes. HPC3 needs to be serviced every 51.2 us to satisfy the receiver channel.

ETHERNET DMA TRANSFER



ETHERNET PROGRAMMED I/O TRANSFER



2.5 SCSI Interface

HPC3 will have two identical but independent ports which will support the Western Digital WD33C93, WC33C95 and the Fujitsu 86603 SCSI controllers. Each port will have one DMA channel and one programmed I/O channel. The interface to the SCSI device can be either 8 or 16 bits. The programmed I/O interface is expected to be 8 bits. The DMA interface may be 8 or 16 bits.

2.5.1 SCSI Transfer Rates

The maximum rate of the WD33C93B SCSI controller is 10 Mbytes/s (8 bits). The maximum rate of the WD33C95 and Fujitsu 86603 controllers is 20 Mbytes/s (16 bits). HPC3 may not always achieve these maximum transfer rates. HPC3 is designed to be synchronous to the ***gio_clk***. The minimum number of clock cycles for each DMA transfer is 2.5. With a ***gio_clk*** at 25 MHz, this is a maximum transfer rate of 10 Mbytes/s (20 Mbytes/s with 16 bits). If ***gio_clk*** is 33 MHz, the number of clock cycles each DMA transfer is likely to be 3.5, which is a transfer rate of 9.52 Mbytes/s (19 Mbytes/s with 16 bits).

2.5.2 Memory structure and DMA transfers

DMA descriptors are used to control main memory transfers. Both transmission and reception may have more than one DMA descriptor. SCSI blocks generally range from 512 bytes to 64k bytes.

The processor starts the DMA operation by writing the **ch_active** bit in the HD control register. Upon detecting that **ch_active** is active, HPC3 will request a GIO64 cycle to read the first DMA descriptor from main memory and reset the fifo pointers for this channel. The **dir** bit in the HD control register tells the HPC3 the direction of transfer (**dir**=1 is transmit, **dir**=0 is receive). After getting the first DMA descriptor, HPC3 will proceed with the transfer.

If the SCSI channel is receiving data (**dir**=0), the 3 low bits of the fifo pointers are loaded with the 3 low bits from the 1st ***cbp*** (current buffer pointer) fetched. This means that the channel will not accept any data until the 1st DMA descriptor is fetched (HPC3 needs to where to start putting data into its fifo buffer). This is to allow any alignment for the data buffer in main memory. However, each subsequent descriptor must be aligned to its predecessor.

The DMA operation is halted by writing the **ch_reset** bit in the HD control register. When **ch_reset** becomes active, all data in the fifo is invalidated and all transfer activity stopped. HPC3 asserts ***hd_reset_n*** to reset the SCSI controller and the **ch_active** bit is cleared. The DMA operation can also be stopped by clearing the **ch_active** bit.

****** BUG ******

Currently, HPC3 has a problem with the end of a descriptor chain. When receiving bytes from the SCSI controller, HPC3 will refuse to take the last byte (or bytes if in 16 bit mode). There is a way to make HPC3 behave correctly. When receiving, always tack on an extra DMA descriptor to the end of the chain. If the bytecount in the extra DMA descriptor is zero, there are no extra bytes transferred, and HPC3 will merrily transfer all bytes.

2.5.2.1 Transmit Operation

HPc3 will request the gio64 bus for data transfers whenever the three high order bits of the fifo byte count (the number of bytes the fifo has room for) is greater than or equal to the high water mark (3 bit value in the DMA configuration register). HPC3 also requests the GIO64 bus whenever the current byte count (number of bytes left to transfer in the main memory buffer) is less than the fifo byte count.

When the current byte count is zero, HPC3 will fetch the next DMA descriptor unless the **eox** flag is set. The **eox** flag notifies HPC3 that this is the last descriptor to process. As soon as HPC3 has processed this descriptor, it stops reading from main memory. When the transfer to the SCSI device is complete, HPC3 clears the **ch_active** bit.

2.5.2.2 Receive Operation

HPC3 will request the GIO64 bus for a data transfer whenever the 3 high order bits of the fifo byte count (number of bytes in the fifo) is greater than or equal to the high water mark. HPC3 will also request the gio64 bus when the current byte count (space left in the main memory buffer) is less than the fifo byte count.

When the current byte count is zero, HPC3 will fetch the next DMA descriptor unless the **eox** flag is set. The **eox** flag notifies HPC3 that this is the last descriptor to process.

When receiving data, descriptors are processed until either the **eox**=1 flag is encountered or the **flush** bit (HD control register) is enabled. After the device transfers all its data to the HPC3, it interrupts the host, and the host will program the **flush** bit. When this bit is programmed, HPC3 will transfer all the data in its buffer regardless of the state of its high water mark. HPC3 then clears the **ch_active** bit, and HPC3 will wait for the next **ch_active** active. Alternatively, when HPC3 sees the **eox**=1 flag, it goes through the same process as if the **flush** bit were programmed.

2.5.3 Interrupts

There are two sources of interrupts for the HD DMA channel. One is a PIO parity error (checking enabled by **pio_parity_en** in the PIO configuration register) and the other is processing a DMA descriptor whose XIE flag has been set.

The PIO parity interrupt is generated as soon as the bad parity is detected on the SCSI interface (HPC3 checks for odd parity). There is a **parity_error** status bit in the control register which can be read. This status bit is cleared whenever the control register is read. This status bit is shared by the PIO and DMA parity functions (**parity_error** can be set through a DMA or a PIO parity error). The DMA parity error does not trigger an interrupt.

The xie interrupt activated by the DMA descriptor **xie**=1 flag. This results in an interrupt generated when HPC3 is finished processing the DMA descriptor. The interrupt is always generated as soon as HPC3 is finished with the main memory buffer which the descriptor described. When transmitting, this means that the bytes being transferred may still be in HPC3's fifo buffer, waiting to be sent to the SCSI controller. The interrupt is cleared when the control register is read.

2.5.4 16 bit operations

HPC3 supports a 16 bit interface to the Fujitsu 86603 controller. Both 16 bit DMA and 16 bit PIO operations are possible. There are seperate controls (enables) for DMA and PIO.

2.5.4.1 DMA

dma_16 (lives in the ***dma configuration*** register) enables 16 bit DMA operations. When doing 16 bit DMA transfers, the total number of bytes transferred must be an even number. If receiving (device to main memory) data, there is a further restriction put on the DMA descriptors. The first DMA descriptor in a chain must be on a half word boundary (start on an even address). Note that this implies that the last byte in a transfer will occupy an odd address (total number of bytes transferred must be even).

If we really want to transfer an odd number of bytes to the controller, each case (transmit and receive) must be looked at seperately. When transmitting, the number of bytes described by the descriptor chain is one more than the actual number to transmit (an even number). HPC3 will transfer all of its bytes to the Fujitsu controller. The Fujitsu controller has been programmed with the correct number of bytes and will toss the extra byte received from HPC3. When receiving, the number of bytes described by the descriptor chain is one less than the actual number to receive (again, an even number). HPC3 will transfer all of its bytes to main memory. The last byte must be fetched from the Fujitsu controler through programmed I/O (PIO).

The 16 bit interface between HPC3 and the Fujitsu controller is a little endian bus. This should be transparent to software. However, if there is a need, there exists a way to swap the bytes within the half word. **dma_swap** (lives in the ***dma configuration*** register) is the enable for this feature and affects both incoming (receive) and outgoing (tranmit) data.

2.5.4.2 PIO

16 bit PIO transfers are enabled by the **pio_16** (lives in the ***pio configuration*** register). As all HPC3 PIO registers are considered word quantities, regardless if we are in big or little endian mode, bits 15:0 are sent directly to the Fujitsu controller. However, when **pio_swap** (lives in the ***pio configuration*** register) is enabled, bits 15:8 and 7:0 are swapped for both incoming (read) and outgoing (write) data.

It is expected that all PIO operations will be 8 bits.

2.5.5 Implementation

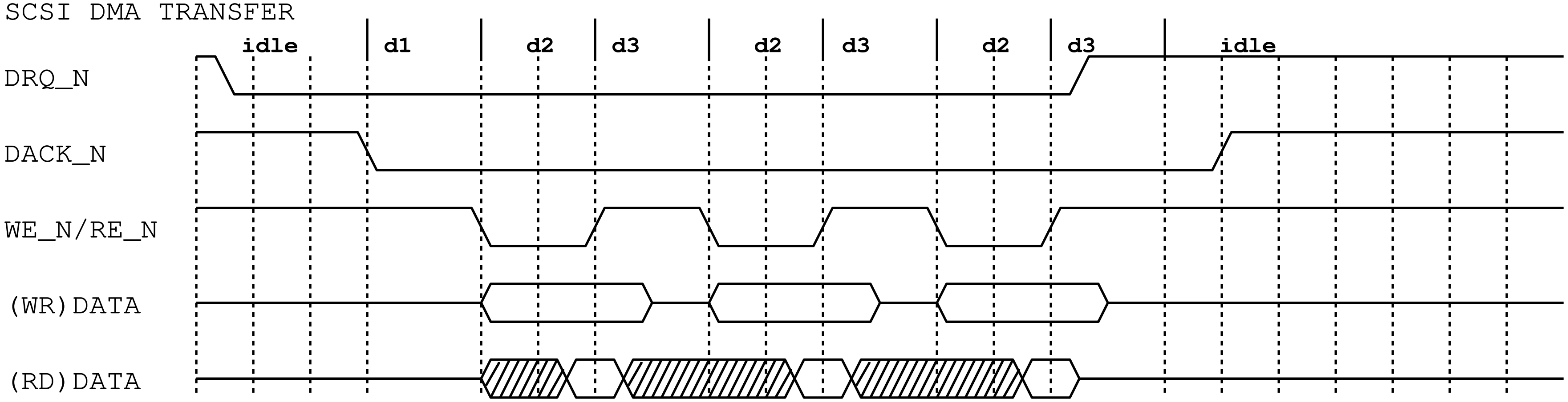
There SCSI main memory state machine handshakes with the GIO64 bus state machine to transfer data between the SCSI fifo buffer and main memory. The SCSI channel state machine controls the transfers between the SCSI device and the SCSI fifo buffer. The channel state machine also handshakes with the GIO64 bus state machine to do programmed I/O with the SCSI device control registers.

2.5.5.1 Fifo Buffer

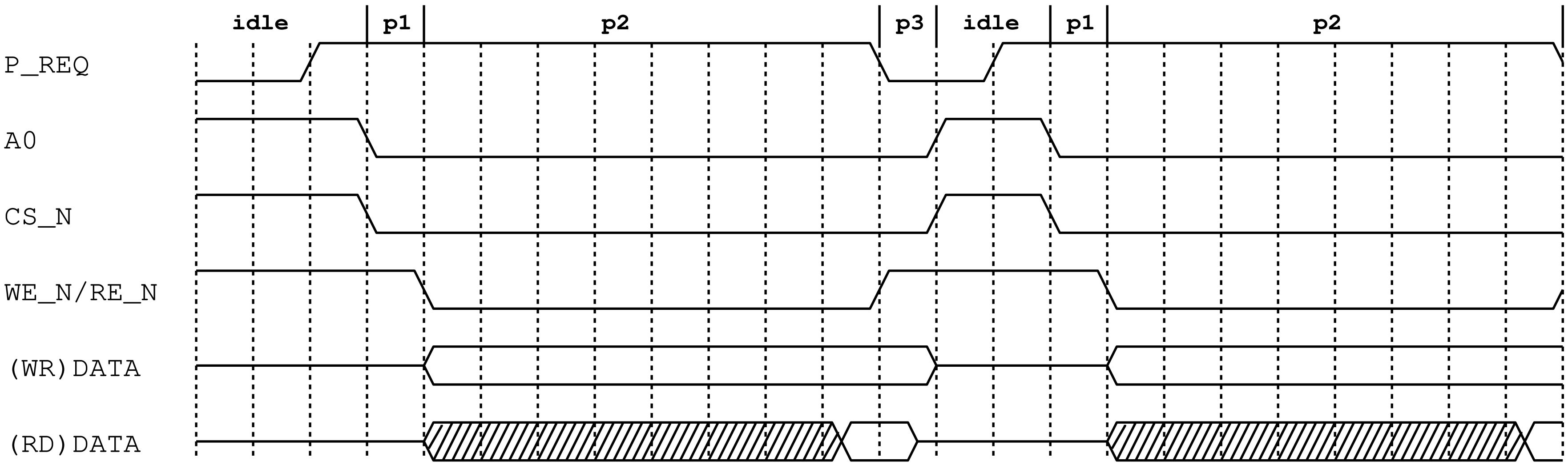
The SCSI fifo buffer has 4 ports, 2 read ports and 2 write ports. The gio side has 1 read and 1 write port. The device side has 1 read and 1 write port. During a DMA write operation, the main memory state machine (on the gio side) will be using it’s write port and the channel state machine (on the device side) will be using it’s read port. During a DMA read operation, the channel state machine will be using it’s write port and the main memory state machine will be using it’s read port. Although each port is independent, during a DMA operation HPC3 should never be accessing the same location in the fifo from both sides simultaneously. During a DMA operation, the fifos should never be accessed through programmed I/O.

The channel state machine must not overflow/underflow the fifo buffer. It is impossible for the main memory state machine to overflow/underflow the fifo buffer.

SCSI DMA TRANSFER



SCSI PROGRAMMED I/O TRANSFER



2.6 PBUS Controller

The PBUS is a general purpose I/O peripheral bus, designed to support several attached low bandwidth devices. There is special support for the Boot PROM address space and a 64K segment of battery backed ram. There is general support for 10 programmed I/O channels and 8 DMA channels. The PBUS controller is designed to support a maximum aggregate DMA bandwidth of 8 Mbytes/second. There is a 16 bit data bus, and each PIO and DMA channel can be configured to service either an 8 or 16 bit device. For 8 bit devices, the position on the bus (high or low) is also configured - so that loading can be reduced. For 16 bit devices, the position on the bus (high or low) of the even address bytes is configured - this will allow support for a byte swap if needed. Eight bit transfers with 16 bit devices are not supported. There is a 20 bit address bus. All address bits are used when addressing the PROM. The 16 least significant address bits are used when addressing the battery backed ram. For the general purpose PIO channels, an 8 bit address is used. This address is copied onto address bits (15:8) and (7:0) to reduce loading. There is an active low read strobe and an active low write strobe used for both PIO and DMA operations. There is also a general purpose registered output signal which can be used as a status indication or a hardware reset.

To support multiple PIO and DMA channels, a simple arbitration scheme is used. DMA channels can be configured as *real_time* or *general*. General DMA channels will typically support long burst DMA devices which will transfer several bytes, but are able to tolerate long latencies from request to service. General DMA channels are also able to tolerate preemption of a DMA burst by the HPC3. Real_time DMA channels will typically support devices which transfer a smaller number of bytes, but can not tolerate long latencies. Real_time DMA channels are not preemptable by HPC3 unless the HPC3 runs out of data in the case of transmit, or fills its fifo in the case of receive. PIO and DMA will alternately win the bus when both require service. This will prevent many consecutive PIO writes from *hoarding* the bus and blocking service to real_time DMA channels. It is also required to prevent real_time DMA from *hoarding* the bus and causing the GIO BUS to timeout while waiting for PIO. General DMA channels can be preempted at any time in the middle of a burst by any PIO or real_time DMA channel. There is also a configurable *burst_count* for general DMA channels. After performing burst_count consecutive transfers, a general DMA channel can be preempted by another general DMA channel. In the event of a tie between one or more DMA channels, the least significant channel wins, e.g., ch_1 and ch_5, ch_1 gets the PBUS.

The PBUS has been designed in such a way that it should be easy to connect devices which only require PIO, or devices whose DMA interface matches the Intel 8237 DMA controller interface.

2.6.1 PBUS Programmed I/O

The PBUS controller supports the general PIO model described in section 2.1. Each channel has a set of configuration registers which specify the timing characteristics of the attached device. There are two dedicated Boot PROM chip selects that each decode a 2 Mbyte address space. There is one dedicated battery backed ram chip select which decodes a 64 Kword segment.

The 10 general purpose chip selects each decode 256 word segments. Recall that the devices are actually 8 or 16 bits wide, but each device register is mapped into a word address in the system. The Boot PROM has a special organization which will be described below.

2.6.2 PBUS Direct Memory Access

The PBUS controller supports the general DMA model described in section 2.2. Each channel has a set of configuration registers which specify the timing characteristics of the attached device. There is an active low DMA request/acknowledge pair for each of the 8 DMA channels. There is one active high terminal count signal for all PBUS DMA channels. The TC output signal will be asserted in state D4 of the device DMA cycle for the last transfer of an operation, i.e., EOX set and byte_count satisfied. If the TC signal is required at the end of a DMA read operation, there is a constraint on the size of the last main memory buffer in the descriptor chain. The last buffer in main memory must be larger than the HPC3 fifo for the given DMA channel. If the TC signal is required at the end of a DMA write operation, then the last main memory buffer in the descriptor chain must have at least one byte.

For all PBUS DMA channels, there is one 384 byte ram. It is organized as a 48 word by 64 bit, byte writtable memory. It is 64 bits wide so that GIO64 transfers can proceed without stalls. Each channel will use a section of the ram as its fifo. The sections can be of different sizes so that the fifo size can be matched to the bandwidth of the attached device - fifos must be some multiple of 8 bytes long and are doubleword aligned. Each channel will have some registers to maintain its fifo. The fifo_beg is the doubleword address of the beginning of the ram section for a given channel. The fifo_end is the doubleword address of the end of the block. These are 6 bit values ranging from 0 to 47. The device side and GIO64 side each interface to the fifo, and need to know the address of the next byte to be read or written. These byte addresses are the gio_ptr and gen_ptr respectively. They are 10 bit values where the nine least significant bits range from 0 to 383, and the most significant bits toggle when the pointer wraps around from the end of the fifo to the beginning. Each channel also needs a high_water mark. This indicates when GIO64 service should be requested. For DMA reads, when there are high_water bytes *used* in the fifo, the channel will request service. For DMA writes, when there are high_water bytes *unused* in the fifo, the channel will request service.

If for some reason there is the possibility of there being extra bytes in the HPC3 fifo after the completion of a DMA read operation, the software can detect this and recover the data. The gio_ptr and gen_ptr can be read from the HPC3 and compared. If they are not equal, the fifo has stray bytes. Since the PBUS ram is in the PIO address space, the ram can be read using PIO to recover the data.

2.6.3 PBUS Devices

It is important to note that except for the Boot PROM and battery backed ram, the PBUS controller does not know which devices are connected. Some of the devices that may be connected are: Boot PROM, audio part, interrupt controller, real time clock, floppy disk controller, parallel port, and serial port. The following sections give some information on these devices.

2.6.3.1 Boot PROM

There is special support for two 16 bit PROMs. Each PROM can be up to 2 Mbytes. For PROM PIO, block transfers (including CPU Subblock Ordering) are *not* supported. The PROM can be accessed for reads of byte, halfword, word, and doubleword data sizes. Each system access, therefore, may require multiple device accesses to construct the necessary data to be transferred. Byte reads are actually no different from halfword reads. The HPC3 determines how much PROM data to read based on the three least significant bits of the GIO bus Byte Count. If these three bits are 1 or 2, then a halfword is read. If these three bits are 3 or 4, then a word is read. If these three bits are 5, 6, 7 or 0, then a doubleword is read. The PROM is organized in words. Even PROM addresses access bits (31:16) of a word, and odd PROM addresses access bits (15:0) of a word. This will allow a piece of code that is in the PROM to be executed by a big or little endian process since bits, not bytes, are read from the same location for a big or little endian process. HPC3 can be configured to perform writes to the PROM in order to provide support for future use of a FLASH PROM.

2.6.3.2 Audio

HPC3 will interface to the HAL2 chip for audio DMA. Audio DMA is expected to use 4 real_time DMA channels. This might be extended to six in systems where the DMA channels are available. More details about this interface will be provided.

2.6.3.3 Interrupt Controller, INT2

The INT2 interrupt controller will be used to collect all of the interrupts in the machine. INT2 then sends six interrupt bits to MC which issues a write to the R4000 to cause an interrupt. The EISA interrupts will be collected in an EISA chip and then sent to INT2. INT2 will use one of the PBUS chip selects.

2.6.3.4 Real Time Clock

At least two different real time clock chips can be used. The first is a Dallas DS1286. The second part is a Dallas DS1386 which is the same part except that it can have up to 32 Kbytes of battery backed ram in it. There is a special chip select for the part with 32 Kbytes of battery backed ram. When the special chip select is used all 16 bits of the PBUS address bus will be valid.

2.6.4 PBUS Bandwidth

The following is a discussion of PBUS bandwidth. The main purpose of the PBUS is to provide DMA support for the HAL2 chip of the A2 audio system. The audio DMA requires timely service in the sense that the audio fifos in HAL2 can Overflow/Underflow if the PBUS does not provide adequate service. This is by definition a failure. Two questions are addressed here. Do we guarantee audio performance? How much bandwidth is left over for other PBUS devices?

HPC3 registers: (see HPC3 Chip Specification for details)
----

pbus.cfgdma(i) should be set to 0x08248844 for the audio DMA channels. This will set the timing correctly for HAL2 and configure the channels as real_time on the PBUS.

The en_real_time bit in the gio.misc register should be set if the machine uses the gio_timeout_n feature.

The real_time bits of the pbus.ctrl(i) registers for the audio DMA channels should be set if the machine uses the gio_timeout_n feature.

Audio sample period and sample sizes:

The audio transfer rate is 48 KHz, so the sample period is 20 us.

Currently audio uses four DMA channels -- PBUS(0:3). The sample sizes of the four channels are {64 128 128 192} bits, or {8 16 16 24} bytes.

HAL2 is a 16 bit PBUS device, so the number of transfers per sample is {4 8 8 12}.

The burst DMA timing for HAL2 is 4 cycles per transfer, plus 3 cycles per burst of PBUS overhead. This gives a transfer time per sample of {19 35 35 51} cycles.

For 25 MHz GIO these transfer times are {0.76 1.4 1.4 2.04} us.

HPC3 fifo sizes and highwater marks:

Each of the four audio fifos should be sized to hold exactly four samples. The highwater mark should be set to two samples.
fifo sizes: {4 8 8 12} doublewords (rows of the PBUS fifo)
highwater: {8 16 16 24} halfwords (highwater expressed in halfwords)

The parallel port fifo should be sized to about 32 bytes (4 rows)
with the highwater mark about half way (8 halfwords).

Audio buffers in main memory:

Audio buffers in main memory always have an integral number of samples. They are also hundreds of samples long.

PBUS PIO overhead and the PBUS arbiter:

We need to ensure that PIO operations get serviced within a reasonable amount of time. Therefore, PIO must be allowed to get on the PBUS while DMA requests are active. However, PIO can not be given absolute priority, or it would be possible to never get a DMA cycle. The PBUS arbiter has a very simple alternating scheme to ensure that both PIO and DMA get on the PBUS. The absolute worst case (though it may be unrealistic) is that there is always a PIO write pending. Therefore, each DMA would be preceded by a PIO. Because of this, PIO transfer times must be kept under control. Currently the maximum time allowed for PIO is 480 ns. This corresponds to 16 cycles at 33.3 MHz or 12 cycles at 25 MHz. Devices which are too slow should not be used; or at least they change this analysis.

The worst case PBUS behavior looks like:
PIO DMA PIO DMA PIO DMA PIO DMA PIO DMA PIO DMA ...

HPC3 on the GIO64 Bus:

ASSUME 10 us from request to grant!!!

HPC3 uses 2 cycles to initiate each gio transaction once it owns the bus.

Reads from memory have 6 (maybe more?) stalls from BYTECOUNT to first DATA.

DMA Descriptor Fetches take 12 cycles
(2 + ADDRESS + BYTECOUNT + 6 + DATA + DATA) = 12
DMA Descriptors are quadrupleword aligned.

DMA Reads have 11 cycles of OVERHEAD
(2 + ADDRESS + BYTECOUNT + 6 + 1) = 11
The extra cycle assumes that the first piece of data is in the second "half" of a quadrupleword, so there is one more stall to get to the next quadrupleword.

Time for enet rcve dma write (write to memory is probably faster):
enet rcve fifo has 16 doublewords, 8 of which can be written to memory in one gio burst
(11 + 8) = 19

Time for enet xmit dma read:
enet xmit fifo has 20 doublewords, 10 of which can be filled from memory in one gio burst
(11 + 10) = 21

PI1 DMA timing:

PBUS overhead of 3 cycles/burst. Transfer time of 7 cycles/xfer.
Assume PI1 DMA is preempted. Therefore, 10 cycles/byte for parallel port DMA (worst case). For 25 MHz GIO this is 0.4 us.

****
* Is audio performance guaranteed? *
****

The following statement has been blessed by the OS people ...
... i.e., Wiltse

PROM and BBRAM PIO shall not happen when the system is using audio in any real mode. Therefore, PROM and BBRAM PIO do not influence the PBUS bandwidth analysis.

The analysis is an attempt to make the least significant audio channel {PBUS(3)} fail. If someone can find a condition worse than this one they shall receive a great reward ($0.25/gio_clk).

To simplify the description xmit DMA will be used as the example.
Rcve DMA has the same characteristics. HAL2 has a one sample buffer on the host side (PBUS side). At time T=0us HAL2 transfers the sample from the host side to the device side and issues a request for PBUS DMA service. This will repeat periodically at the sample rate. Notice that we have approximately one sample period to respond to this request since the HAL2 has the sample for this period already transferred to the device side.

To get the worst case we need to get the GIO bus involved. The highwater mark is reached with this sample, and by the time we get the GIO bus there is lots of work to do.

At T=0us we have just less than three samples in the PBUS fifo (2+).
We have one sample on the device side of HAL2 and 2+ samples in the HPC3. At T=40us data which (as of T=0us) is not yet in the PBUS fifo will be requested by HAL2. This sample must be delivered before T=60us.

The following numbers all assume 25 MHz GIO clock. The times are expressed in micro seconds.

0.4	- about ten clocks of HAL2 request delay
0.4	- preempt parallel port DMA
0.48	- PIO

```
 0.76       - DMA on PBUS(0)
 0.48       - PIO
 1.4        - DMA on PBUS(1)
 0.48       - PIO
 1.4        - DMA on PBUS(2)
 0.48       - PIO
 2.04       - DMA on PBUS(3)
 0.4        - PBUS(3) has reached the highwater mark
              about ten clocks of HPC3 request delay
10.0        - HPC3 request to grant on GIO64 bus
---------
18.72       - HPC3 now on GIO64 bus
 2.88       - descriptor fetches on PBUS(4:7) and SCSI(0:1)
              6 * 12
 3.72       - data transfer on ENET(rcve:xmit) and PBUS(0:2)
              19 + 21 + (3 * 11) + 4 + 8 + 8
 2.4        - descriptor fetches on ENET(rcve:xmit) and
PBUS(0:2)
              5 * 12
 3.24       - data transfer on ENET(rcve:xmit) and PBUS(0:2)
              19 + 21 + (3 * 11) + 2 + 3 + 3
 0.92       - data transfer on PBUS(3)
              11 + 12
 0.24       - about six clocks for data to be ready
---------
32.12       - PBUS(3) ready to DMA to HAL2 now
```

The new sample will be requested at about T=40us.

From this analysis two other observations can be made:

- Partial sample DMA between HPC3 and HAL2 will not occur.
- When HAL2 issues the DMA request, the PBUS fifo is ready for service -- the priority of the PBUS channel determines the latency, which we have already seen is less than 10 us.

This has been a very conservative analysis. By being more aggressive it can be shown that audio performance is guaranteed even when the highwater marks are set to full. In this case we save the second set of GIO transmit DMAs for PBUS(0:2) -- this is 41 cycles or 1.64 us. In addition, PBUS(0:2) either request the GIO bus on their own behalf sooner than PBUS(3), or they do not interfere with PBUS(3) on the PBUS just prior to T=40us. The DMA for PBUS(3) would complete before T=40us.

At 33.3 MHz these numbers get a lot better!

```
*************************************************************
* How much bandwidth is left over for other PBUS devices? *
*************************************************************
```

With the GIO64 bus running at 25 MHz, the audio interface on the PBUS uses about 5.6 us every 20 us on average. This leaves about

14 us for the parallel port. This is more than enough for 1 MB/sec bandwidth. If we consider the worst case when there is an infinite supply of PIO then the audio DMA needs {5.6 + (4 * 0.48)} us. This leaves 12.48 us out of 20 us for other devices. If the PIO continues then parallel port DMA would be one byte at a time with a byte time of (0.48 + 0.4) us. This affords a transfer rate of 14 Bytes/20 us, or 700 KB/sec bandwidth. One great thing about PI1 is that it allows for smooth performance degradation when the PBUS does not provide adequate service. This is because PI1 was implemented to prevent fifo overflow/underflow. It simply stalls the parallel port.

2.7 Serial EEPROM Interface

The EEPROM module is a very simple interface to a National, 93CS56 serial EEPROM. The interface in HPC3 is a 5 bit register. It is up to software to read and write this register to generate the correct waveforms to the EEPROM. There is a serial input from the eeprom (si, read only), a serial output to the eeprom (so), an active high chip select (cs), the protect register enable (pre), and the serial eeprom clock (sck). See the HPC1 specification for a description of how to program the EEPROM by writing and reading this control register.

3.0 HPC3 Address Map

There is space in the system memory map for two HPC3 chips. It is doubtful that a machine would ever be built with two HPC3 chips, but if one wanted to build a machine with lots of I/O devices, (2 ethernet controllers, multiple high speed SCSI channels, FDDI, ISDN, etc.) it would be possible. The address map for the two chips is as follows:

0x1fb00000	0x1fb7ffff	Second HPC3 chip, I/O space
0x1fb80000	0x1fbfffff	First HPC3 chip, I/O space
0x1fc00000	0x1fdfffff	First HPC3 chip, PROM 0 space
0x1fe00000	0x1fffffff	First HPC3 chip, PROM 1 space

There is no PROM space on the second HPC3 chip. The 4 Mbytes of PROM space is evenly split between the two PBUS PROM chip selects.

The 512 Kbytes of I/O space for each HPC3 is divided into nine sections. For the first chip, they are as follows:

0x1fb80000	0x1fb8ffff	PBUS DMA channel registers
0x1fb90000	0x1fb9ffff	HD0, HD1, ENET DMA channel registers
0x1fba0000	0x1fbaffff	Fifo access ports
<u>0x1fbb0000</u>	<u>0x1fbbffff</u>	<u>General registers</u>
0x1fbc0000	0x1fbc7fff	HD0 device registers
0x1fbc8000	0x1fbcffff	HD1 device registers
0x1fbd0000	0x1fbd7fff	ENET device registers
<u>0x1fbd8000</u>	<u>0x1fbdffff</u>	<u>PBUS device registers</u>
0x1fbe0000	0x1fbfffff	Battery backed sram address space

The address space for the first HPC3 chip is as follows:

Register Name	Address	Read/Write	Description
pbus.bp(0)	0x1fb80000	R	pbus dma channel 0, buffer pointer
pbus.dp(0)	0x1fb80004	R/W	pbus dma channel 0, descriptor pointer
pbus.ctrl(0)	0x1fb81000-0x1fb81fff	R/W	pbus dma channel 0, control register
pbus.dmareg(1)	0x1fb82000-0x1fb83fff	R/W	pbus dma channel 1, dma registers same structure as above bp, dp, and ctrl
pbus.dmareg(2)	0x1fb84000-0x1fb85fff	R/W	pbus dma channel 2, dma registers bp. dp, and ctrl
pbus.dmareg(3)	0x1fb86000-0x1fb87fff	R/W	pbus dma channel 3, dma registers bp, dp, and ctrl
pbus.dmareg(4)	0x1fb88000-0x1fb89fff	R/W	pbus dma channel 4, dma registers bp, dp, and ctrl
pbus.dmareg(5)	0x1fb8a000-0x1fb8bfff	R/W	pbus dma channel 5, dma registers bp, dp, and ctrl
pbus.dmareg(6)	0x1fb8c000-0x1fb8dfff	R/W	pbus dma channel 6, dma registers bp, dp, and ctrl
pbus.dmareg(7)	0x1fb8e000-0x1fb8ffff	R/W	pbus dma channel 7, dma registers bp, dp, and ctrl
hd0.cbp	0x1fb90000	R	SCSI channel 0, cbp
hd0.nbdp	0x1fb90004	R/W	SCSI channel 0, nbdp

hd0.bc	0x1fb91000	R	SCSI channel 0, bc
hd0.cntl	0x1fb91004	R/W	SCSI channel 0, control
hd0.gio	0x1fb91008	R	SCSI channel 0, gio fifo ptr
hd0.dev	0x1fb9100c	R	SCSI channel 0, device fifo ptr
hd0.dmacfg	0x1fb91010	R/W	SCSI channel 0, DMA configuration
hd0.piocfg	0x1fb91014	R/W	SCSI channel 0, PIO configuration
hd1.cbp	0x1fb92000	R	SCSI channel 1, cbp
hd1.nbdp	0x1fb92004	R/W	SCSI channel 1, nbdp
hd1.bc	0x1fb93000	R	SCSI channel 1, bc
hd1.cntl	0x1fb93004	R/W	SCSI channel 1, control
hd1.gio	0x1fb93008	R	SCSI channel 1, gio fifo ptr
hd1.dev	0x1fb9300c	R	SCSI channel 1, device fifo ptr
hd1.dmacfg	0x1fb93010	R/W	SCSI channel 1, DMA configuration
hd1.piocfg	0x1fb93dd014	R/W	SCSI channel 1, PIO configuration
enetr.cbp	0x1fb94000	R	ethernet receiver, cbp
enetr.nbdp	0x1fb94004	R/W	ethernet receiver, nbdp
enetr.bc	0x1fb95000	R	ethernet receiver, bc
enetr.cntl	0x1fb95004	R/W	ethernet receiver, control
enetr.gio	0x1fb95008	R	ethernet receiver, gio fifo ptr
enetr.dev	0x1fb9500c	R	ethernet receiver, device fifo ptr
enet.reset	0x1fb95014	R/W	ethernet, reset
enet.dmacfg	0x1fb95018	R/W	ethernet, DMA configuration
enet.piocfg	0x1fb9501c	R/W	ethernet, PIO configuration
enetx.cbp	0x1fb96000	R	ethernet transmitter, cbp
enetx.nbdp	0x1fb96004	R/W	ethernet transmitter, nbdp
enetx.bc	0x1fb97000	R	ethernet transmitter, bc
enetx.cntl	0x1fb97004	R/W	ethernet transmitter, control
enetx.gio	0x1fb97008	R	ethernet transmitter, gio fifo ptr
enetx.dev	0x1fb9700c	R	ethernet transmitter, device fifo ptr
pbus.fifo	0x1fba0000- 0x1fba7fff	R/W	pbus fifo address range
hd0.fifo	0x1fba8000- 0x1fba9fff	R/W	hd0 fifo address range
hd1.fifo	0x1fbaa000- 0x1fbabfff	R/W	hd1 fifo address range
enet.rfifo	0x1fbac000- 0x1fbadfff	R	ethernet, receiver fifo address range
enet.xfifo	0x1fbae000- 0x1fbaffff	W	ethernet, transmitter fifo address range
gen.intstat	0x1fbb0000	R	interrupt status, bits 4:0
gen.intstat.bug	0x1fbb000c	R	interrupt status, bits 9:5
gio.misc	0x1fbb0004	R/W	gio64 bus, misc
eeprom.data	0x1fbb0008	R/W	serial eeprom data register
gio.bus_error	0x1fbb0010	R/W	gio64 bus error interrupt status
hd0.cs	0x1fbc4000-	R/W	SCSI channel 0, external registers

	0x1fbc43ff		
hd1.cs	0x1fbcc000- 0x1fbcc3ff	R/W	SCSI channel 1, external registers
enet.cs	0x1fbd4000- 0x1fbd44ff	R/W	ethernet, external registers
pbus.pio(0)	0x1fbd8000- 0x1fbd83ff	R/W	pbus pio channel 0, external registers
pbus.pio(1)	0x1fbd8400- 0x1fbd87ff	R/W	pbus pio channel 1, external registers
pbus.pio(2)	0x1fbd8800- 0x1fbd8bff	R/W	pbus pio channel 2, external registers
pbus.pio(3)	0x1fbd8c00- 0x1fbd8fff	R/W	pbus pio channel 3, external registers
pbus.pio(4)	0x1fbd9000- 0x1fbd93ff	R/W	pbus pio channel 4, external registers
pbus.pio(5)	0x1fbd9400- 0x1fbd97ff	R/W	pbus pio channel 5, external registers
pbus.pio(6)	0x1fbd9800- 0x1fbd9bff	R/W	pbus pio channel 6, external registers
pbus.pio(7)	0x1fbd9c00- 0x1fbd9fff	R/W	pbus pio channel 7, external registers
pbus.pio(8)	0x1fbda000- 0x1fbda3ff	R/W	pbus pio channel 8, external registers
pbus.pio(9)	0x1fbda400- 0x1fbda7ff	R/W	pbus pio channel 9, external registers
pbus.pio(8)	0x1fbda800- 0x1fbdabff	R/W	pbus pio channel 8, external registers
pbus.pio(9)	0x1fbdac00- 0x1fbdafff	R/W	pbus pio channel 9, external registers
pbus.pio(8)	0x1fbdb000- 0x1fbdb3ff	R/W	pbus pio channel 8, external registers
pbus.pio(9)	0x1fbdb400- 0x1fbdb7ff	R/W	pbus pio channel 9, external registers
pbus.pio(8)	0x1fbdb800- 0x1fbdbbff	R/W	pbus pio channel 8, external registers
pbus.pio(9)	0x1fbdbc00- 0x1fbdbfff	R/W	pbus pio channel 9, external registers
pbus.cfgdma(0)	0x1fbdc000- 0x1fbdc1ff	R/W	pbus dma channel 0, configuration register
pbus.cfgdma(1)	0x1fbdc200- 0x1fbdc3ff	R/W	pbus dma channel 1, configuration register
pbus.cfgdma(2)	0x1fbdc400- 0x1fbdc5ff	R/W	pbus dma channel 2, configuration register
pbus.cfgdma(3)	0x1fbdc600- 0x1fbdc7ff	R/W	pbus dma channel 3, configuration register
pbus.cfgdma(4)	0x1fbdc800- 0x1fbdc9ff	R/W	pbus dma channel 4, configuration register
pbus.cfgdma(5)	0x1fbdca00- 0x1fbdcbff	R/W	pbus dma channel 5, configuration register
pbus.cfgdma(6)	0x1fbdcc00- 0x1fbdcdff	R/W	pbus dma channel 6, configuration register
pbus.cfgdma(7)	0x1fbdce00-	R/W	pbus dma channel 7, configuration

	0x1fbdcfff		register
pbus.cfgpio(0)	0x1fbdd000- 0x1fbdd0ff	R/W	pbus pio channel 0, configuration register
pbus.cfgpio(1)	0x1fbdd100- 0x1fbdd1ff	R/W	pbus pio channel 1, configuration register
pbus.cfgpio(2)	0x1fbdd200- 0x1fbdd2ff	R/W	pbus pio channel 2, configuration register
pbus.cfgpio(3)	0x1fbdd300- 0x1fbdd3ff	R/W	pbus pio channel 3, configuration register
pbus.cfgpio(4)	0x1fbdd400- 0x1fbdd4ff	R/W	pbus pio channel 4, configuration register
pbus.cfgpio(5)	0x1fbdd500- 0x1fbdd5ff	R/W	pbus pio channel 5, configuration register
pbus.cfgpio(6)	0x1fbdd600- 0x1fbdd6ff	R/W	pbus pio channel 6, configuration register
pbus.cfgpio(7)	0x1fbdd700- 0x1fbdd7ff	R/W	pbus pio channel 7, configuration register
pbus.cfgpio(8)	0x1fbdd800- 0x1fbdd8ff	R/W	pbus pio channel 8, configuration register
pbus.cfgpio(9)	0x1fbdd900- 0x1fbdd9ff	R/W	pbus pio channel 9, configuration register
pbus.cfgpio(8)	0x1fbdda00- 0x1fbddaff	R/W	pbus pio channel 8, configuration register
pbus.cfgpio(9)	0x1fbddb00- 0x1fbddbff	R/W	pbus pio channel 9, configuration register
pbus.cfgpio(8)	0x1fbddc00- 0x1fbddcff	R/W	pbus pio channel 8, configuration register
pbus.cfgpio(9)	0x1fbddd00- 0x1fbddddff	R/W	pbus pio channel 9, configuration register
pbus.cfgpio(8)	0x1fbdde00- 0x1fbddeff	R/W	pbus pio channel 8, configuration register
pbus.cfgpio(9)	0x1fbddf00- 0x1fbddfff	R/W	pbus pio channel 9, configuration register
pbus.prom_we	0x1fbde000- 0x1fbde7ff	W	pbus boot prom write enable register
pbus.prom_swap	0x1fbde800- 0x1fbdefff	W	pbus boot prom chip select swap register
pbus.gen_out	0x1fbdf000- 0x1fbdffff	W	pbus general purpose output register
pbus.bbram	0x1fbe0000- 0x1fbfffff	R/W	pbus battery backed ram external registers

3.1 GIO64 Bus Interface Control Registers

intstat This is the DMA interrupt status register for all DMA channels except for Ethernet. This is a read only register. Reading this register has no effect on the status bits. The status bits can also be read in their local control/status registers.

(7:0) **PBUS interrupt status**
(8) **SCSI channel 0 interrupt status**
(9) **SCSI channel 1 interrupt status**

***** BUG *****
There is a problem with the read back of this register. Instead of being in one piece, it is broken and can only be read in two pieces. Bits 4:0 can be read from 0x1fbb0000. Bits 9:5 can be read from 0x1fbb000c. All other bits in both registers should be ignored as they are indeterminate.

misc Miscellaneous collection of bits.

(0) **en_real_time** Enable for the real time feature with the external timer. **en_real_time** allows the real time devices to stay on the bus as long as they need service.

(1) **des_endian** DMA descriptor endian orientation. **des_endian**=1 is little endian.

bus_error This is the bus error interrupt status register. This only kind of bus error HPC3 checks for is parity. If a parity (HPC3 checks for even parity on the gio64 bus) error occurs, information about what caused the error can be read here. The bus error interrupt is reset upon a read of this register.

***** BUG *****
Ideally, there should be another status bit in this register to indicate whether a parity error actually occurred. If the ***bus_error_intr*** pin is logically ored with another interrupt pin on the board, there is no way to determine from HPC3 if it was the cause of the interrupt. To get around this, software should service HPC3 last. If no other device caused the interrupt, then HPC3 must be the culprit. This will work as long as the other devices which share the interrupt behave properly (are able to indicate whether they caused the interrupt).

(7:0) **byte_lane_err** Holds the byte lane status. A ’0’ indicates good parity.

(8)	**PIO/DMA**	PIO/DMA indicates the type of cycle during which the parity error occured. A '0' means the parity error happened during a PIO cycle.
(18:9)	**parity_id**	For a DMA parity error, **parity_id** is the DMA channel identifier. Bits 18:9 represent channels 11:0 respectively. Only one bit should be enabled (='1') as a bit is enabled when the DMA **parity error** occured during the channel's DMA cycle. During a PIO parity error, **parity_id** is the PIO process identifier. Bits 18:9 are bits 22:11 of the PIO address.

3.2 Ethernet Registers

3.2.1 gio_clk registers

rx_cbp This is the current buffer pointer for the receiver channel and is part of the DMA descriptor. ***rx_cbp*** points to the main memory buffer for the current DMA transfer. ***rx_cbp*** should only be updated through DMA descriptor fetches (and not through PIO).

rx_nbdp This is the next buffer descriptor pointer for the receiver channel and is part of the DMA descriptor. ***rx_nbdp*** points to the next DMA descriptor in the chain. When starting a DMA operation, ***rx_nbdp*** should be programmed to point to the first descriptor in the chain.

rx_bc This is the byte count information for the receiver channel which is part of the DMA decriptor.

(13:0)	**des_bc**	Amount of bytes to tranfer to the main memory buffer. This field is updated during a descriptor fetch and during a data transfer. This field is a read only field for PIO accesses.
(29)	**xie**	Flag to tell HPC3 to interrupt the host after HPC3 is done transferring data to the current main memory buffer. This bit should always be set (=1 is interrupt). This bit is a read/write bit for PIO accesses.
(31)	**eox**	Flag to tell HPC3 that the current descriptor is the end of the descriptor chain (=1 is end of chain). This bit is a read/write bit for PIO accesses.

rx_control This is the control register which directs the actions of this DMA channel.

(5:0)	**status_5_0**	Read only interrupt status. This is a copy of the Receive Status Register in the Seeq 8003 controller.
(6)	**status_6**	Read only interrupt status. If this bit is a '1' and the other status bits are '0', then a late_rxdc condition occurred (***rxdc*** arrived after 64 bytes were received by HPC3). If this bit is a '1' and status_5_0 bit 4 is a '1'

		then a timeout condition occured. Either HPC3 received an **eop** without the corresponding interrupt or HPC3 received the eop interrupt without an eop.
(7)	**status_7**	Read only old/new status. This is a copy of the bit in the receive status register in the SEEQ 8003 controller.
(8)	**endian**	Endian mode for this DMA channel. Little endian when **endian** = ’1’ and bit endian when **endian** = ’0’. This bit is a read/write bit for PIO accesses.
(9)	**ch_active**	Indicates a DMA transfer in progress. When ch_active = ’1’, this DMA channel is active, when **ch_active** = ’0’, this DMA channel is inactive. To start a DMA transfer, all other parameters should be programmed before writing **ch_active** to a ’1’. HPC3 will reest **ch_active** to ’0’ when the transfer is complete. This bit is a read/write bit for PIO accesses. This bit can only be written when **ch_active_mask** is a ’0’. This bit is reset (= ’0’) upon power-on reset.
(10)	**ch_active_mask**	When writing to the control port and **ch_active_mask** = ’1’, writes are inhibited to **ch_active**, and ch_active will retain it’s current value; when **ch_active_mask** = ’0’, writes are enabled to **ch_active**, and **ch_active** will be updated. This bit is a write only bit for PIO accesses.
(11)	**rbo**	**rbo** = 1 (receiver buffer overflow) indicates that the incoming packet was larger than the main memory buffer allocated for it. This is a read only bit.

rx_gio This is the gio fifo pointer (the fifo pointer which is used when putting bytes in the main memory buffer). The gio fifo pointer is the fifo tail. This port is a read/write port for PIO accesses. In normal operation of this chip, this port should never be written..

rx_dev This is the device fifo pointer (the fifo pointer which is used when collecting bytes from the SEEQ)

The device fifo pointer is the fifo head. This port is a read/write port for PIO accesses. In normal operation of this chip, this port should never be written.

misc This collects miscellaneous bits together which don't quite fit in any other place.

(0)	**ch_reset**	Channel reset. Resets both external controller and this DMA channel. This bit is active (=1, channel is reset) upon power-on reset. This must be programmed to a 0 before the **ch_active** bit becomes active. This bit is a read/write bit for PIO accesses.
(1)	**clrint**	Clear channel interrupt. Writing a "1" to this bit will clear all ethernet interrupts except for rbo (rbo is cleared by a read from the control register where it lives). Reading this bit gives the current status of the ethernet interrupt.
(2)	**loopback**	Enables loopback mode for the Seeq 8003. This bit goes straight out to the SEEQ. This is a diagnostic feature.

pio_cfg This is the PIO configuration register.

(3:0)	**pio_p1**	Number of ***gio_clk*** cycles to spend in state P1 for external PIO accesses. The number of clocks is from 1 to 16. This value is set to "1111" (16) upon power-on reset. Refer to the Ethernet timing diagram for PIO accesses.
(7:4)	**pio_p2**	Number of ***gio_clk*** cycles to spend in state P2 for external PIO accesses. The number of clocks is from 1 to 16. This value is set to "1111" (16) upon power-on reset. Refer to the Ethernet timing diagram for PIO accesses.
(11:8)	**pio_p3**	Number of ***gio_clk*** cycles to spend in state P3 for external PIO accesses. The number of clocks is from 1 to 16. This value is set to "1111" (16) upon power-on reset. Refer to the Ethernet timing diagram for PIO accesses.
(12)	**test_ram**	Enables a special test mode for the transmitter fifo ram. The eop byte in

			the ram can be written by programmed I/O. The ram should be written in two steps. The eop byte is first written, always at xxx. Then the ram is written. This is for testing only, and should always be inactive during normal operation of this chip.
dma_cfg		This is the DMA configuration register.	
	(3:0)	**dma_d1**	Number of ***gio_clk*** cycles to spend in state D1 for DMA accesses to/from the device. The number of clocks is from 1 to 16. This value is set to "1111" (16) upon power-on reset. This is currently not implemented, and the value is always 1. Refer to the Ethernet timing diagram for DMA accesses.
	(7:4)	**dma_d2**	Number of ***gio_clk*** cycles to spend in state D2 for DMA accesses to/from the device. The number of clocks is from 1 to 16. This value is set to "1111" (16) upon power-on reset. Refer to the Ethernet timing diagram for DMA accesses.
	(11:8)	**dma_d3**	Number of ***gio_clk*** cycles to spend in state D3 for DMA accesses to/from the device. The number of clocks is from 1 to 16. This value is set to "1111" (16) upon power-on reset. Refer to the Ethernet timing diagram for DMA accesses.
	(12)	**wr_ctrl**	Enables writing a byte from the DMA descriptor into the extended control port (interpacket gap timing) of the SEEQ 8003.
	(13)	**fix_rxdc**	Enables clearing eop status bits (eop_in_chip and rcv_eop_intr) upon rxdc. This is to fix a bug in the SEEQ chip)
	(14)	**fix_eop**	Enables a timeout counter to start counting whenever eop_in_chip is set (HPC3 has received the eop bit from SEEQ). If HPC3 has not received the corresponding interrupt (which sets rcv_eop_intr) before the counter expires, HPC3 marks this packet bad, and sends it to main memory tagged as such.

(15) **fix_intr** Enables a timeout counter to start counting whenever rcv_eop_intr is set (HPC3 has received the end of packet interupt from SEEQ). If HPC3 has not received the corresponding eop (which sets eop_in_chip) before the counter expires, HPC3 marks this packet bad, and sends it to main memory tagged as such.

(17:16) **pgm_timeout** This is to allow some flexibility in the timeout counter mentioned above. Both features use the same counter.

tx_cbp This is the current buffer pointer for the transmitter channel and is part of the DMA descriptor. **tx_cbp** points to the main memory buffer for the current DMA transfer. **tx_cbp** should only be updated through DMA descriptor fetches (and not through PIO). This is a read/write port.

tx_nbdp This is the next buffer descriptor pointer for the transmitter channel and is part of the DMA descriptor. **tx_nbdp** points to the next DMA descriptor in the chain. When starting a DMA operation, **tx_nbdp** should be programmed to point to the first descriptor in the chain. This is a read/write port.

tx_bc This is the byte count information for the transmitter channel and is part of the DMA decriptor.

(13:0) **des_bc** Amount of bytes to tranfer from main memory buffer. This field is updated during a descriptor fetch and during a data transfer. This field is a read only field for PIO accesses.

(28) **eox_sampled** Indicates whether it is too late to add a descriptor to the end of the chain. HPC3 samples **eox** when **eop** is written into the transmitter fifo. **eox_sampled** becomes active when eox is '1' when **eop** is written into the fifo. **eox_sampled** is cleared when **ch_active** is '0'. This is a readback bit only.

(29) **xie** Flag to tell HPC3 to interrupt the host after HPC3 is done transferring data to the current descriptor buffer. HPC3 will interrupt the host after the timeout counter has expired. This bit is a read/write bit for PIO accesses.

(30)	**eop**	Flag to tell HPC3 that the last byte in this buffer is end of the current packet.
(31)	**eox**	Flag to tell HPC3 that the current descriptor is the end of the descriptor chain. This bit is a read/write bit for PIO accesses.

tx_control This is the control register which directs the actions of this DMA channel.

(3:0)	**status_3_0**	Read only interrupt status. This is a copy of the Transmit Status Register in the Seeq 8003 controller.
(4)	**status_4**	Read only interrupt status. If this bit is a '1', a late collision has occured (HPC3 receives the late collision interrupt after 80 bytes has been transferred to the SEEQ controller).
(7:5)	**status_7_5**	Read only interrupt status. This is a copy of bit 7 of the Transmit Status Register in the Seeq 8003 controller.
(8)	**endian**	Endian mode for this DMA channel. Little endian when **endian** = '1' and bit endian when **endian** = '0'. This bit is a read/write bit for PIO accesses.
(9)	**ch_active**	Indicates a DMA transfer in progress. When ch_active = '1', this DMA channel is active, when **ch_active** = '0', this DMA channel is inactive. To start a DMA transfer, all other parameters should be programmed before writing **ch_active** to a '1'. HPC3 will turn **Ch_active** to a '0' when the transfer is complete. This bit is a read/write bit for PIO accesses. This bit can only be written when **ch_active_mask** is a '0'. This bit is reset (= '0') upon power-on reset.
(10)	**ch_active_mask**	When writing to the control port and **ch_active_mask** = '1', writes are inhibited to **ch_active**, and ch_active will retain it's current value; when **ch_active_mask** = '0', writes are enabled to **ch_active**, and **ch_active** will be updated. This bit is a write only bit for PIO accesses.

tx_gio This is the gio fifo pointer (the fifo pointer which is used when fetching bytes from the main memory buffer). The gio fifo pointer is the fifo head. This port is a read/write port for PIO accesses. In normal operation of this chip, this port should never be written..

tx_dev This is the device fifo pointer (the fifo pointer which is used when sending bytes to the SEEQ chip). The gen fifo pointer is the fifo tail. This port is a read/write port for PIO accesses. In normal operation of this chip, this port should never be written.

3.3 SCSI Registers

3.3.1 gio_clk registers

cbp This is the current buffer pointer and is part of the DMA descriptor. ***cbp*** points to the main memory buffer for the current DMA transfer. ***cbp*** should only be updated through DMA descriptor fetches during normal operation of this chip. This is a read/write port for PIO accesses.

nbdp This is the next buffer descriptor pointer and is part of the DMA descriptor. ***nbdp*** points to the next DMA descriptor in the chain. When starting a DMA operation, ***nbdp*** should be programmed to point to the first descriptor in the chain. When trying to add on to the descriptor chain while HPC3 is working on the last descriptor, ***nbdp*** should be updated before updating the **eox** bit. This is a read/write port for PIO accesses.

bc This is the byte count information and is part of the DMA decriptor.

(13:0)	**des_bc**	Amount of bytes to tranfer to/from main memory buffer. This field is updated during a descriptor fetch and during a data transfer. This field is a read only field for PIO accesses.
(29)	**xie**	Flag to tell HPC3 to interrupt the host after HPC3 is done transferring data to the current descriptor buffer. HPC3 will interrupt the host after the last byte has been transferred to/from main memory. This bit is a read/write bit for PIO accesses.
(31)	**eox**	Flag to tell HPC3 that the current descriptor is the end of the descriptor chain. This bit is a read/write bit for PIO accesses.

control This is the control register which directs the actions of this DMA channel.

(0)	**interrupt**	Read only interrupt status bit (interrupt cleared on read of this port). This status bit can also be read back from the general interrupt port, which has no effect on the

interrupt. This bit is set when a dma interrupt occurs or when a parity error is detected on the SCSI interface.

(1)	**endian**	Endian mode for this DMA channel. Little endian when **endian** = '1' and bit endian when **endian** = '0'. This bit is a read/write bit for PIO accesses.
(2)	**dir**	Direction of transfer for this DMA channel. From main memory to device when **dir** = '1' and from device to main memory when **dir** = '0'. This bit is a read/write bit for PIO accesses.
(3)	**flush**	Tells HPC3 to "flush" contents of it's fifos to main memory. This should only be programmed when channel is programmed to receive (**dir** = '0'). Note that an interrupt does not occur automatically when the flush is complete. It depends on whether the **XIE** bit is set for the last descriptor it was working on. This bit is a read/write bit for PIO accesses.
(4)	**ch_active**	Indicates a DMA transfer in progress. When ch_active = '1', this DMA channel is active, when **ch_active** = '0', this DMA channel is inactive. To start a DMA transfer, all other parameters should be programmed before writing **ch_active** to a '1'. HPC3 will turn **ch_active** to a '0' when the transfer is complete. This bit is a read/write bit for PIO accesses. This bit can only be written when **ch_active_mask** is a '0'. This bit is reset (= '0') upon power-on reset.
(5)	**ch_active_mask**	When writing to the control port and **ch_active_mask** = '1', writes are inhibited to **ch_active**, and ch_active will retain it's current value; when **ch_active_mask** = '0', writes are enabled to **ch_active**, and **ch_active** will be updated. This bit is a write only bit for PIO accesses.
(6)	**ch_reset**	Channel reset. Resets both external controller and this DMA channel. This bit is active (=1, channel is reset) upon power-on reset. This must be programmed to a 0 before the **ch_active**

bit becomes active. This bit is a read/write bit for PIO accesses.

(7) **parity_error** Indicates parity error detected on interface to SCSI controller. This flag is cleared upon a read of this port. This can be a DMA parity error (if **dma_parity_en** = '1') or a PIO parity error (if **pio_parity_en** = '1'). A pio parity error (on the SCSI interface) will generate an interrupt. **dma_parity_en** and **pio_parity_en** are in the ***dma_cfg*** and ***pio_cfg*** registers respectively.

gio This is the gio fifo pointer (the fifo pointer which is used when transferring data to/from the main memory buffer). When the channel is a receiver (**dir** = '0'), the gio fifo pointer is the fifo tail. When the channel is a transmitter (**dir** = '1'), the gio fifo pointer is the fifo head. This port is a read/write port for PIO accesses. In normal operation of this chip, this port should never be written..

dev This is the device fifo pointer (the fifo pointer which is used when transferring data to/from the scsi controller). When the channel is a receiver (**dir** = '0'), the gen fifo pointer is the fifo head. When the channel is a transmitter (**dir** = '1'), the gen fifo pointer is the fifo tail. This port is a read/write port for PIO accesses. In normal operation of this chip, this port should never be written.

pio_cfg This is the device configuration register for pio accesses.

(1:0) **pio_p3** Number of ***gio_clk*** cycles to spend in state P3 for external PIO accesses. The number of clocks is from 1 to 4. This value is set to "00" (1) upon power-on reset. Refer to the SCSI timing diagrams for PIO acceses.

(4:2) **pio_p2_wr** Number of ***gio_clk*** cycles to spend in state P2 for external PIO write accesses. The number of clocks is from 1 to 8. This value is set to "000" (1) upon power-on reset. Refer to the SCSI timing diagrams for PIO accesses.

(8:5)	**pio_p2_rd**	Number of ***gio_clk*** cycles to spend in state P2 for external PIO write accesses. The number of clocks is from 1 to 16. This value is set to "0000" (1) upon power-on reset. Refer to the SCSI timing diagrams for PIO accesses.
(11:9)	**pio_p1**	Number of ***gio_clk*** cycles to spend in state P1 for external PIO accesses. The number of clocks is from 1 to 8. This value is set to "000" (1) upon power-on reset. Refer to the SCSI timing diagrams for PIO accesses.
(12)	**pio_16**	This enables 16 bit pio accesses to the SCSI controller. 16 bit accesses are enables when **pio_16** = '1'.
(13)	**pio_swap**	Enables byte swap during PIO accesses. Outgoing PIO data is swapped as well as incoming PIO data. The data will be swapped if **pio_swap**=1. This bit is reset upon power-on reset.
(14)	**pio_parity_en**	Enables parity checking for pio data transfers between HPC3 and the SCSI controller. A pio parity error will generate an interrupt (on the ***dma_intr*** pin) and set a status bit in the control port (**parity_error**). HPC3 checks for and generates odd parity. (HPC3 always generates parity).
(15)	**fuji_mode**	Enables a faster switch from dma to pio cycles on the SCSI interface.

dma_cfg This is the device configuration register for dma accesses.

(0)	**half_clock**	Puts DMA state machine into half clock mode.
(2:1)	**dma_d1**	Number of ***gio_clk*** cycles to spend in state D1 for DMA accesses to/from the device. The number of clocks is from 1 to 8. This value is set to "000" (1) upon power-on reset. This is currently not implemented, and the value is always 1. Refer to the SCSI timing diagram for DMA accesses.
(5:3)	**dma_d2**	Number of ***gio_clk*** cycles to spend in state D2 for DMA accesses to/from the

		device. The number of clocks is from 1 to 8. This value is set to "000" (1) upon power-on reset. Refer to the SCSI timing diagram for DMA accesses.
(8:6)	**dma_d3**	Number of ***gio_clk*** cycles to spend in state D3 for DMA accesses to/from the device. The number of clocks is from 1 to 8. This value is set to "000" (1) upon power-on reset. Refer to the SCSI timing diagram for DMA accesses.
(11:9)	**hwm**	This is the high water mark, which control when this DMA channel will request the GIO64 bus. First, HPC3 calculates the number of bytes to transfer (based on the current fifo pointers, the direction of transfer, and the current descriptor byte count), the three high order bits of this number is then compared to these three bits. If the **hwm** is less than or equal to the number of bytes to transfer, then this DMA channel will request the gio64 bus. Note that the **hwm** should never be set to "111" as this is a value larger than the fifo size. This value is set to "100" upon power-on reset.
(12)	**dma_16**	When **dma_16** = '1', 16 bit dma transfers are done. When **dma_16** = '0', 8 bit dma transfers are done.
(13)	**dma_swap**	When **dma_swap** = '1', a byte swap is performed on the half word dma data. This bit is valid only when **dma_16** = '1'.
(14)	**dma_parity_en**	Enables parity checking for dma data transfers. A parity error on a DMA transfer does not generate an interrupt. There is a status bit (**parity_error**) in the control port which indicates whether a parity error (dma or pio) occurred. HPC3 checks for and generates odd parity. (HPC3 always generates parity).
(15)	**dreq_pol**	Polarity control for output ***hd_dreq_n***. If **dreq_pol** = '0', ***hd_dreq_n*** is active low, if **dreq_pol** = '1' then ***hd_dreq_n*** is active high.
(17:16)	**dreq_early**	Selects between four versions of ***hd_dreq_n***. "00" and "01" selects synchronized versions of ***hd_dreq_n***. "00" selects falling rising edge sync flops, "01" selects rising falling edge sync

flops. "10" selects ***hd_dreq_n*** flopped once. "11" selects unflopped ***hd_dreq_n***.

3.4 PBUS Registers

All PBUS registers are described below. Recall that all registers are addressed as 32 bit words regardless of their actual size. Bit fields are indicated numerically and symbolically wherever applicable.

pbus.pio(i) and pbus.bbram are external registers, and each word address within their address ranges accesses a unique external register. pbus.cfgdma(i), pbus.cfgpio(i), pbus.prom_we, pbus.prom_swap, pbus.gen_out, and the pbus.dmareg(i) registers are all internal registers of the HPC3 chip. Although Section 3.0 indicates a range of addresses for each of these registers, each is just one single register. For example, any word address in the range for pbus.cfgdma(0) will access the same register.

pbus.pio(i) address space for the ten external PBUS PIO channels. Each PIO channel decodes 256 word registers. Note that PIO channels 2 through 7 are repeated in the address map (Section 3.0). This avoids having any 'holes' in the address space.

pbus.bbram battery backed ram address space. The battery backed ram PIO channel decodes 32K words of external static ram. The actual ram will likely be 8 bits wide.

pbus.cfgdma(i) configuration registers for the eight PBUS DMA channels. For D4 and D5 cycle counts, 0 means 2**4=16 cycles. 32 bits are kept internally.

0	**rd_d3**	number of ***gio_clk*** cycles to spend in DMA state D3 for DMA read operations. Two(0) or three(1) cycles.
(4:1)	**rd_d4**	number of ***gio_clk*** cycles to spend in DMA state D4 for DMA read operations.
(8:5)	**rd_d5**	number of ***gio_clk*** cycles to spend in DMA state D5 for DMA read operations.
9	**wr_d3**	number of ***gio_clk*** cycles to spend in DMA state D3 for DMA write operations. Two(0) or three(1) cycles.
(13:10)	**wr_d4**	number of ***gio_clk*** cycles to spend in DMA state D4 for DMA write operations.
(17:14)	**wr_d5**	number of ***gio_clk*** cycles to spend in DMA state D5 for DMA write operations.
18	**ds_16**	data size 16. Active high indicates a 16 bit device.
19	**even_high**	active high indicates for a 16 bit device that the even address bytes are on the high portion of the PBUS data bus (15:8). For an 8 bit device, active high indicates that the device is connected to PBUS data bits (15:8).
20	**unused**	not used.
21	**real_time**	active high indicates a *real_time* DMA device.
(26:22)	**burst_count**	5 bit *burst_count* for *general* DMA device.
27	**drq_live**	use the live (unsynchronized)

pbus_dreq_n(i) input for the trailing edge to indicate DMA preemption by the device. For most devices this is possible, but if the device does not have a clean drq signal, then it must be synchronized - even for the trailing edge.

pbus.bp(i) buffer pointers for the eight PBUS DMA channels.

pbus.dp(i) descriptor pointers for the eight PBUS DMA channels.

pbus.ctrl(i) control registers for the eight PBUS DMA channels. These registers have completely different meaning for read compared with write.

pbus.ctrl(i) write

1	**little**	little endian DMA transfer.
2	**receive**	type of DMA is receive.
3	**flush**	enable flush for receive DMA.
4	**ch_act**	channel active bit (start DMA).
5	**ch_act_ld**	load enable for **ch_act**. If **ch_act_ld** is a 1 **ch_act** will be written. Otherwise, **ch_act** will not be written. This allows the **flush** bit to be written without affecting the **ch_act** bit.
6	**real_time**	enable *real time* GIO bus service. This affects HPC3 behavior on the GIO bus. See section 2.3.3 for details.
(15:8)	**highwater**	trigger level for GIO bus service. This value is expressed in bytes / 2, or, equivalently, in halfwords. A highwater mark of 22 bytes would be indicated by a value of 11 in this bit field.
(21:16)	**fifo_beg**	offset of first row in PBUS fifo ram for the channel i fifo (0 to 47).
(29:24)	**fifo_end**	offset of last row in PBUS fifo ram for the channel i fifo (0 to 47).

pbus.ctrl(i) read

0	**interrupt**	interrupt signal (cleared after read).
1	**ch_act**	channel active bit.

pbus.cfgpio(i) configuration registers for the ten PBUS PIO channels. Note that the configuration registers for channels 8 and 9 are repeated in the address map (Section 3.0). This avoids having any 'holes' in the address space. For P3 and P4 cycle counts, 0 means 2**4=16 cycles. 32 bits are kept internally.

0	**rd_p2**	number of ***gio_clk*** cycles to spend in PIO state P2 for PIO read operations. One(1) or two(0) cycles.
(4:1)	**rd_p3**	number of ***gio_clk*** cycles to spend in

		PIO state P3 for PIO read operations.
(8:5)	**rd_p4**	number of ***gio_clk*** cycles to spend in PIO state P4 for PIO read operations.
9	**wr_p2**	number of ***gio_clk*** cycles to spend in PIO state P2 for PIO write operations. One(1) or two(0) cycles.
(13:10)	**wr_p3**	number of ***gio_clk*** cycles to spend in PIO state P3 for PIO write operations.
(17:14)	**wr_p4**	number of ***gio_clk*** cycles to spend in PIO state P4 for PIO write operations.
18	**ds_16**	data size 16. Active high indicates a 16 bit device.
19	**even_high**	active high indicates for a 16 bit device that the even address bytes are on the high portion of the PBUS data bus (15:8). For an 8 bit device, active high indicates that the device is connected to PBUS data bits (15:8).

pbus.prom_we write enable for the boot PROM (write only).

0	**prom_we**	write enable for the boot PROM. Active high, enables writes to both prom0 and prom1. prom_we resets to 0 so that writes are disabled.

pbus.prom_swap swap prom0 and prom1 address spaces (write only).

0	**prom_swap**	active high inverts gio64 address bit 21 which selects between prom0 and prom1. Effectively, the address spaces are swapped. During reset ***pbus_dma_tc*** is an input pin. The value input on this pin is flopped into **prom_swap** during reset. Under normal conditions, ***pbus_dma_tc*** should be pulled low on the board.

pbus.gen_out general purpose output bit (write only).

0	**gen_out**	single general purpose output bit. Used for status indication or hardware reset of attached devices. Resets to 0.

3.5 Serial EEPROM Registers

eeprom.data five bit Serial EEPROM data register.

0	**pre**	eeprom protect register enable
1	**cs**	eeprom chip select
2	**clk**	eeprom clock
3	**dato**	serial data output
4	**dati**	serial data input (read only)

4.0 HPC3 Pins

There are 25x signal pins on the HPC3 array. This includes 4 pins for test, and 4 pins for JTAG. The 304 MQUAD, metal quad flatpack, will be the package used for this part.

4.1 GIO64 Bus Interface Pins

gio_ad(63:0)	i/o	GIO64 multiplexed address and data bus.
gio_adp(7:0)	i/o	Even parity over the gio_ad(63:0) bus. Gio_adp(0) covers gio_ad(7:0).
gio_vld_parity_n	i/o	The GIO64 bus parity signals, gio_adp, are valid.
gio_as_n	i/o	GIO64 bus, address strobe.
gio_read	i/o	GIO64 bus read/write control, and bus-valid signal.
gio_masdly	i/o	GIO64 bus master delay signal.
gio_slvdly	i/o	GIO64 bus slave delay signal.
gio_breq_n	out	GIO64 bus request to the GIO64 bus arbiter.
gio_bgnt_n	in	GIO64 bus grant from the GIO64 bus arbiter.
gio_timeout_n	in	GIO64 bus timeout

4.2 Ethernet Pins

enet_d(8:0)	i/o	Data bus to/from Seeq ethernet controller.
enet_addr(2:0)	out	Ethernet controller address.
enet_rd_n	out	Read strobe (active low).
enet_wr_n	out	Write strobe (active low).
enet_reset_n	out	Active low reset signal.
enet_txwr_n	out	Write strobe for transmit data to controller.
enet_rxrd_n	out	Read strobe for receive data from controller.
enet_loopback	out	Put manchester ENDEC in loopback mode.
enet_txrdy	in	Controller is ready for more transmit data.
enet_rxrdy	in	Controller has receive data ready.
enet_rxdc	in	Discard last received data.
enet_txret	in	Retransmit last transmitted data.
enet_intr_in	in	Interrupt request from controller.
enet_intr_out	out	Interrupt request to interrupt controller (int).

4.3 SCSI/IDE Pins

There are two SCSI/IDE ports on the HPC3 chip. Either port maybe configured to run as a SCSI port or an IDE port. They also can both be configured to be IDE or SCSI. Both ports are identical, except for the signal names and the addresses of the HPC3 registers associated with each port. The pins for port 0 are listed first with a description followed by the list of pins for port 1. The signal description for port 1 signals is the same as port 0.

hd0_d(15:0)	i/o	Data bus to/from IDE interface or SCSI controller.
hd0_dp(1:0)	i/o	Parity on the 16 data bits. dp(0) covers d(7:0).
hd0_addr(4:0)	out	Address bits to the controller.
hd0_rd_n	out	Read strobe (active low).
hd0_wr_n	out	Write strobe (active low).
hd0_cs_n	out	Active low chip select for PIO controller registers.
hd0_reset	out	Active high reset signal to the controller and scsi bus.
hd0_dreq_n	in	DMA request (programmable polarity).
hd0_dack_n	out	DMA acknowledge (active low).
hd1_d(15:0)	i/o	Same as above.
hd1_dp(1:0)	i/o	
hd1_addr(4:0)	out	
hd1_rd_n	out	
hd1_wr_n	out	
hd1_cs_n	out	
hd1_reset	out	
hd1_dreq_n	in	
hd1_dack_n	out	

4.4 PBUS Pins

pbus_data(15:0)	i/o	Data bus to/from local peripherals.
pbus_addr(19:0)	out	Address bus for PBUS peripherals. For PROM PIO all 20 bits are used. For Battery-backed sram PIO bits (15:0) are used. For general purpose PIO bits (15:8) and (7:0) are identical copies of the eight bit external address.
pbus_rd_n	out	Read strobe (active low).
pbus_wr_n	out	Write strobe (active low).
pbus_prom1_cs_n	out	Active low chip select for system prom 1.
pbus_prom0_cs_n	out	Active low chip select for system prom 0.
pbus_bbram_cs_n	out	Active low chip select for Battery-backed sram.
pbus_cs_n(9:0)	out	Active low general purpose chip selects.
pbus_dreq_n(7:0)	in	Active low general purpose dma requests.
pbus_dack_n(7:0)	out	Active low general purpose dma acknowledges.
pbus_dma_tc	out	DMA terminal count to tell DMA slaves that the dma transfer is complete. Active high.
pbus_gen_out(0)	out	General purpose registered output. Can be used for hardware reset, or

status indication.

4.5 Serial EEPROM Pins

eeprom_dati	in	Serial data from eeprom. This is simply an input port from the eeprom's data output pin.
eeprom_dato	out	Serial data/control to eeprom. This is a registered output port to the eeprom's serial data input pin.
eeprom_clk	out	Clock for serial eeprom. This is a registered output port to the eeprom's clock pin.
eeprom_cs	out	Active high chip select. This is a registered output port to the eeprom's chip select pin.
eeprom_pre	out	Protect register enable. This is a registered output port to the eeprom's pre pin.

4.6 Misc Pins

gio_clk	in	GIO64 bus clock. can vary from 25-33mhz to meet current GIO64 specs.
reset_n	in	Active low reset signal.
jtdi	in	Boundary scan serial data input.
jtdo	out	Boundary scan serial data output.
jtms	in	Boundary scan mode select.
jtck	in	Boundary scan clock.
entei		
tp0		
tp1		
pll_vdd		
pll_vss		
pll_agnd		
pll_lp1		
pll_lp2		
pll_reset_n		
hpc1	in	Configure chips address space as HPC 1.
dma_complete_int	out	DMA complete interrupt.
bus_error_int	out	Bus error interrupt.

5.0 Misfeatures!

5.1 SCSI DMA device transfer

Problem with transferring last byte when receiving data from SCSI controller. Workaround is to allocate extra DMA descriptor with a bytecount of zero.

5.2 General Interrupt Register

The interrupt status cannot be read back from one location. Bits 5:0 can be read back at 0x1fb80000, bits 9:6 can be read back at 0x1fb80000.

5.3 PIO reads from DMA descriptor ram

Whenever reading the cbp (current buffer pointer) or nbdp (next buffer descriptor pointer) for any DMA channel, there can be no pending PIO writes. To make sure of this fact, read from any HPC3 register (an on-chip port will be faster) right before reading from the DMA descriptor ram. Please make sure that no interrupt service routine comes between the two reads!

5.4 Bus error interrupt

There is no read back indicator for the bus error interrupt HPC3 generates. Various status bits are provided, but no bit to say whether HPC3 caused the interrupt.

5.5 PBUS fifo ram pio write address

The PBUS fifo ram pio write address is misaligned. The read address comes from ***gio_ad*(8:3)** while the write address comes from ***gio_ad*(9:4)**.

5.6 PROM chip select

The PROM chip selects can toggle near the end of a read. This happens no sooner than two cycles after ***pbus_rd_n*** goes inactive. The chip select for the selected PROM may go inactive, and the chpi select for the other PROM would go active. This should not be a problem. Be sure to use both chip select and read strobe when connecting the PROM. Do not simply ground the output enable (use read strobe) while connecting only the chip select -- use both!

6.0 IO Timing

```
==================================================
= ALL TIMING RELATIVE TO HPC3 GIO_CLK INPUT PIN =
==================================================

These numbers are WCCOM 100 Degrees C.
For BCCOM divide by 4.4.


PBUS_RD_N
PBUS_WR_N
            B8 driver
            clock skew 1ns, clock-to-q 4ns
            B8 intrinsic 2.2ns
            B8 slope 0.072(ns/pf)
            DELAY = [7.2 + (0.072)(Load_pf)] ns

PBUS_ADDR
PBUS_GEN_OUT
            BT8RP driver
            clock skew 1ns, clock-to-q 4ns
            BT8RP intrinsic 3.3ns
            BT8RP slope 0.089(ns/pf)
            DELAY = [8.3 + (0.089)(Load_pf)] ns

PBUS_DATA
            BD8TRPU driver
            clock skew 1ns, clock-to-q 4ns
            BD8TRPU intrinsic 3.3ns
            BD8TRPU slope 0.089(ns/pf)
            DELAY = [8.3 + (0.089)(Load_pf)] ns

PBUS_PROM_CS_N
PBUS_BBRAM_CS_N
PBUS_CS_N
PBUS_DACK_N
            BT4RP driver
            clock skew 1ns, clock-to-q 4ns
            BT4RP intrinsic 3.3ns
            BT4RP slope 0.147(ns/pf)
            DELAY = [8.3 + (0.147)(Load_pf)] ns

PBUS_DMA_TC
            BD8TRP driver
            clock skew 1ns, clock-to-q 4ns
            BD8TRP intrinsic 3.3ns
            BD8TRP slope 0.089(ns/pf)
            DELAY = [8.3 + (0.089)(Load_pf)] ns


PBUS_DATA
            ~5ns buffer
            ~3ns flop setup
            ~1ns clock skew
            ~9ns setup time
```

```
PBUS_DREQ_N
            ~6ns buffer
            ~20ns logic
            ~1ns clock skew
            ~27ns setup time


GIO_AD
GIO_ADP
GIO_VLD_PARITY_N
GIO_AS_N
GIO_READ
GIO_MASDLY
GIO_SLVDLY
            BD8TRPU driver
            clock skew 1ns, clock-to-q 4ns
            BD8TRPU intrinsic 3.3ns
            BD8TRPU slope 0.089(ns/pf)
            DELAY = [8.3 + (0.089)(Load_pf)] ns

            ~2ns buffer
            ~2ns flop setup
            ~1ns clock skew
            ~5ns setup time
            ~1ns hold time


GIO_BREQ_N
ENET_RD_N
ENET_WR_N
ENET_TXWR_N
ENET_RXRD_N
HD0_RD_N
HD0_WR_N
HD1_RD_N
HD1_WR_N
        BT4 driver
            clock skew 1ns, clock-to-q 4ns
            BT4 intrinsic 3.17ns
            BT4 slope 0.116(ns/pf)
            DELAY = [8.17 + (0.116)(Load_pf)] ns

            ~2ns buffer
            ~2ns flop setup
            ~1ns clock skew
            ~5ns setup time
            ~1ns hold time

HD0_DACK_N
HD0_CS_N
HD0_RESET
HD0_ADDR
EEPROM_DATO
EEPROM_CLK
EEPROM_CS
```

```
EEPROM_PRE
ENET_LOOPBACK
ENET_ADDR
HD1_DACK_N
HD1_CS_N
HD1_RESET
HD1_ADDR
BUS_ERROR_INT
DMA_INTR
ENET_INTR_OUT
ENET_RESET_N
            BT4RP driver
            clock skew 1ns, clock-to-q 4ns
            BT4RP intrinsic 3.3ns
            BT4RP slope 0.147(ns/pf)
            DELAY = [8.3 + (0.147)(Load_pf)] ns

            ~2ns buffer
            ~2ns flop setup
            ~1ns clock skew
            ~5ns setup time
            ~1ns hold time

ENET_DATA
            BD4TRPU driver
            clock skew 1ns, clock_to_q 4ns
            BD4TRPU intrinsic 3.3ns
            BT4RPU slope 0.147(ns/pf)
            DELAY = [8.3 + (0.147)(Load_pf)] ns

            ~2ns buffer
            ~2ns flop setup
            ~1ns clock skew
            ~5ns setup time
            ~1ns hold time

HD0_DATA
HD0_DP
HD1_DATA
HD1_DP
            BD4TRPU driver
            clock skew 1ns, clock_to_q 4ns
            BD4TRPU intrinsic 3.3ns
            BT4RPU slope 0.147(ns/pf)
            DELAY = [8.3 + (0.147)(Load_pf)] ns

            ~2ns buffer
            ~2ns flop setup
            ~1ns clock skew
            ~7ns logic in clock path (goes into latch)
            ~-2ns setup time
            ~+6ns hold time

EEPROM_DATI
ENET_INTR_IN
ENET_RXDC
```

ENET_RXRDY
ENET_TXRDY
ENET_TXRET
GIO_TIMEOUT_N
GIO_BGNT_N
RESET_N

~2ns buffer
~2ns flop setup
~1ns clock skew
~5ns setup time
~1ns hold time

HD0_DREQ_N
HD1_DREQ_N

~2ns buffer
~10ns logic in data path
~2ns flop setup
~1ns clock skew
~15ns setup time
~+1ns hold time

Table of Contents

HPC3 Chip Specification

Revision 1.4
January 22, 1993

Lynn Frake
Gil Herbeck
James Tornes

Silicon Graphics Inc.