zx1 mio
(Memory and I/O)

External Reference Specification

1 OPERATION OVERVIEW

This chapter provides an overview of the major zx1 mio blocks and transactions. It is only intended to give the reader a start on understanding the overall architecture and operation of the zx1 mio. The reader is referred to the zx1 ioa ERS for an overview of rope guest functionality.

1.1 zx1-based Systems

Figure 1 shows a block diagram of the zx1 mio in a typical 2-way workstation system. The system is shown in a direct attach DDR main memory configuration.

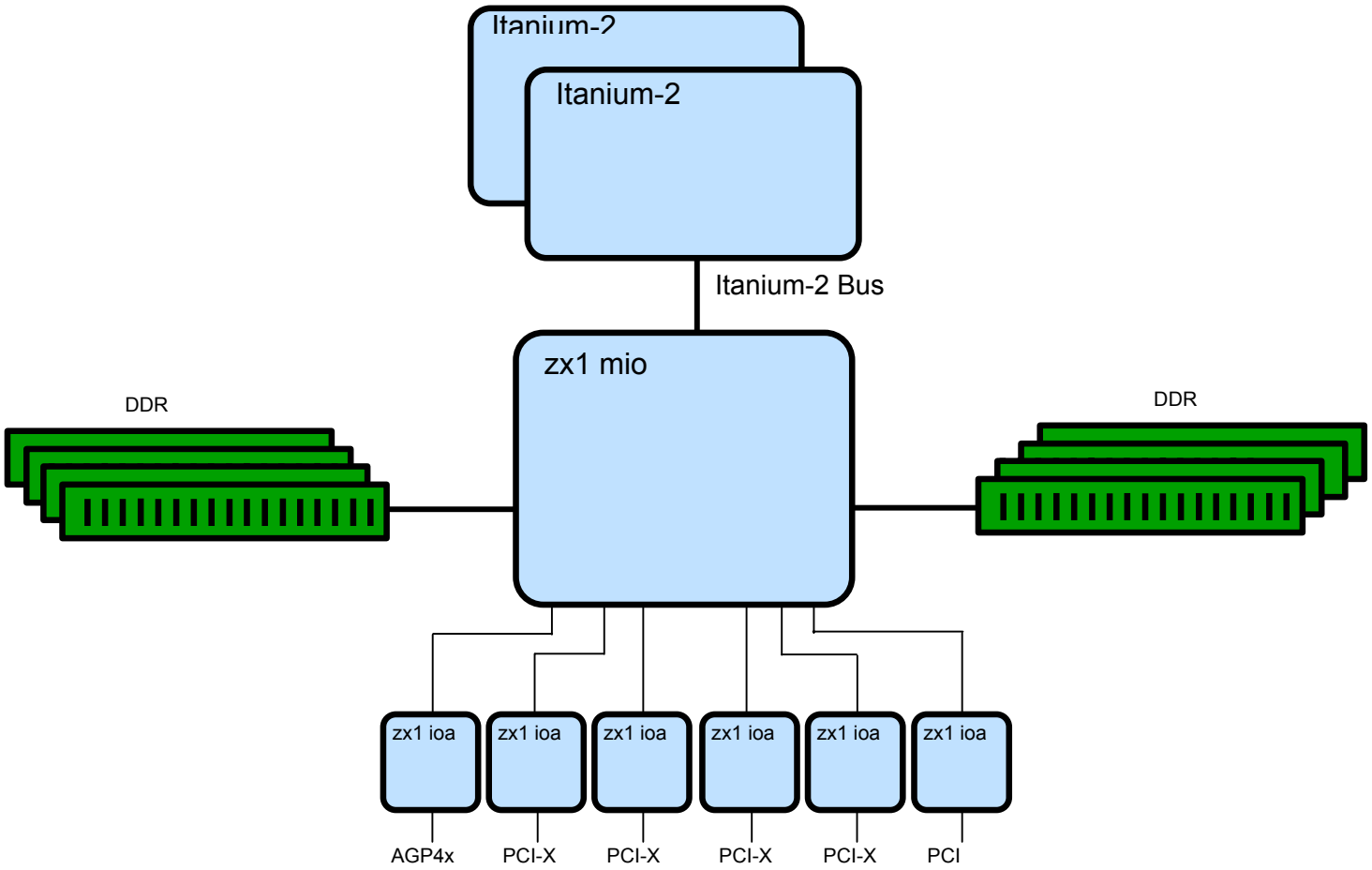

Figure 1: zx1 mio Workstation System Block Diagram

The zx1 mio provides 8 I/O ropes which support PCI, PCI-X, and/or AGP via the zx1 ioa chip. The zx1 mio supports a direct data path between I/O and main memory without crossing the processor bus. The peak I/O bandwidth is 3.2GB/s.

The zx1 mio supports a maximum of 512 DRAM devices (16GB using 256Mb DRAMs) without the use of memory multiplexer chips. This configuration provides a peak bandwidth of 8GB/s in the memory system (w/ 266MHz DDR), capable of extremely low latency (~70nS for idle system open page). The zx1 mio also supports the zx1 sme (Scalable Memory Extender) memory mux chip which will support a maximum of 2048 DDR devices and increase the peak bandwidth to 12.8GB/s.

The Itanium-2 bus will support 1-4 CPU's with a frequency of 200MHz .

1.2 zx1 mio's Major Blocks

The zx1 mio is partitioned into five major blocks (four are unique) as shown in Figure 2. Brief descriptions of each of these blocks are provided below.

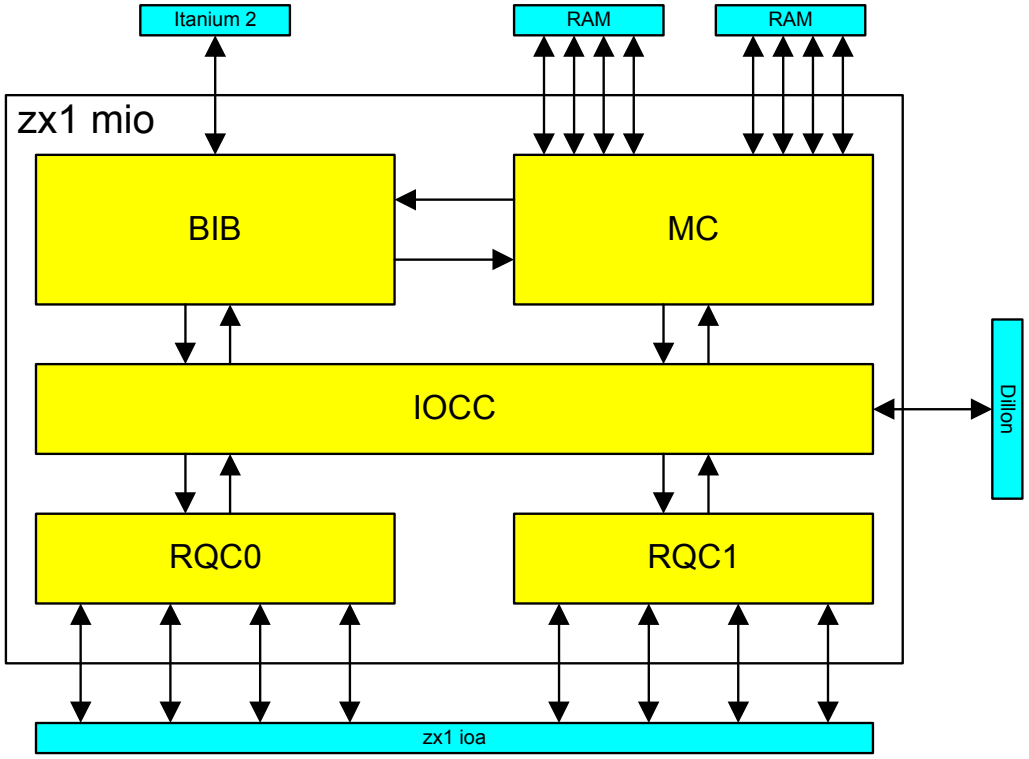

Figure 2: zx1 mio Block Diagram

1.2.1 Itanium-2 bus interface block (BIB)

The BIB block provides the interface between the Itanium-2 bus, the memory controller (MC) and the I/O cache controller (IOCC). Refer to the BIB chapter and the Itanium-2 bus spec for additional Itanium-2 bus information. The pads expand the Itanium-2 data bus from 128-bits externally to 256-bits internally which eliminates the need for any core circuitry to run at 2X the Itanium-2 bus clock. Data is transferred directly between the IOCC and the MC and the Itanium-2 I/O pads. The BIB controls the timing of these transfers and is the only block that understands the Itanium-2 protocol.

The BIB provides the front end processing for the memory controller. It separates transactions into read and write queues. Normally the reads are serviced first but the BIB is responsible for detecting conflicts which will force writes to be executed ahead of reads.

1.2.2 Memory Controller (MC)

The zx1 mio's memory controller provides the interface between the BIB and main memory. The zx1 mio supports main memory in the form of double data rate SDRAM (DDR) directly attached to the zx1 mio chip. This configuration provides the lowest cost and the lowest idle system latency. This configuration will likely be used by any system that can get enough capacity in this fashion. The zx1 mio also supports DDR memory attached via the zx1 sme mux chip. The mux solution will provide four times the memory capacity as the direct attach solution for any given memory technology. The mux configuration also has the advantage of allowing a peak bandwidth that is twice that of the direct attach configuration at the expense of a couple cycles of idle latency.

1.2.3 I/O Cache Controller (IOCC)

The IOCC is responsible for taking DMA requests generated from the RQCs and issuing them to either the Itanium-2 bus or directly to the memory controller. It resolves write requests by maintaining a small (16 entry) coherent cache that is used to merge the I/O write data with existing memory data. DMA read requests are resolved through the re-order buffer which is used to get the data in the proper order and to provide buffering between the Itanium-2 bus data rate and the I/O data rate.

The IOCC also decodes all non-memory transactions and forwards them to their appropriate destination (either one of the ropes, a register, or Processor Dependent Hardware (PDH)).

The IOCC is the home of a 16-entry translation cache (I/O TLB) which uses the IOPDIR in main memory as the source for translating addresses from I/O virtual to physical addresses. The I/O TLB can serve as the GART for AGP graphics devices.

1.2.4 Rope-quad Controller (RQC)

The rope-quad controller provides an interface between the IOCC and up to 4 I/O ropes. The ropes it controls can be bundled as single-wide, double-wide, or quad-wide bundles. The rope-

quad controller provides the majority of "per-rope" functionality such as error detection, Processor I/O Writes / Direct Memory Access Reads (PIOW/DMAR) ordering enforcement, prefetching, flushing, side-band address decoding, etc. The RQC also contains the frequency domain boundary between the Itanium-2 bus clock domain and input and output clock domain for each rope.

1.3 zx1 mio Interconnect

The zx1 mio's primary purpose is a "crossbar" used to move data between several different busses at a very high bandwidth and with very low latency. This section provides brief descriptions of each of those interfaces.

1.3.1 Itanium-2 Bus

Itanium-2 bus is the bus used by the Itanium-2 processors. It has split address and data with a 128-bit data bus. The data bus uses source synchronous clocking to transfer data on both edges of the clock. The Itanium-2 bus is slated to run at 200MHz. The Itanium-2 bus has ECC on the data bus and parity on the address bus. Itanium-2 bus supports up to 50-bits of physical address and has a signal count of approximately 260 signals. Devices on the Itanium-2 bus maintain cache coherency by using a snoopy-based protocol.

1.3.2 I/O Ropes

An I/O rope is a fast, narrow point-to-point connection between the zx1 mio and a rope guest device (the zx1 ioa). The rope guest device is responsible for bridging from the I/O rope to an industry standard I/O bus (PCI, AGP, and PCI-X). I/O ropes can be bundled together to increase the width of the datapath. The zx1 mio has 8 single-wide ropes which can be bundled as double-wide or quad-wide to increase the throughput as appropriate. The zx1 mio supports rope speeds up to 266MHz. At 266MHz a single-wide rope is capable of about 500MB/s peak bandwidth (PCI 4x bandwidth). PDH Bus

1.3.3 PDH Bus

The PDH bus is a 4-bit bus that connects the zx1 mio to the boot ROMs. The PDH bus runs at one fourth of the Itanium-2 bus frequency and provides a point to point interconnect to the Dillon ASIC which in turn interfaces to the PDH ROM, SRAM and other core PDH devices. If Dillon is not present, ROM accesses will be forwarded to I/O rope 0. The control for the PDH bus lives inside the IOCC in the zx1 mio.

1.4 Terminology

1.4.1 Device Terminology

Itanium-2	Definition
Requesting Agent	The device that issues the transaction.
Addressed Agent	The device that is addressed by the transaction.
Responding Agent	The device that provides the response on the RS[2:0]# signals to the transaction. This is typically the addressed agent and will always be zx1 mio.
Snooping Agent	A device that snoops coherent transactions to maintain cache coherency.
Deferring Agent	The device that defers a transaction and accepts responsibility for completing the transaction at a later time.

Table 1: Itanium-2 Device Names

1.4.2 Cache State Terminology

Itanium-2	Definition
Invalid	Cache line is not valid.
Shared	Cache line is valid, not modified and may be in more than one agent's cache.
Exclusive	Cache line is valid, not modified and only in this agent's cache.
Modified	Cache line is valid, modified and only in this agent's cache.

Table 2: Itanium-2 Cache States

1.4.3 Address Space Terminology

Itanium-2	Definition
Memory	Memory address space is a 50-bit address space that is divided into multiple regions. A region may be allocated to either the memory controller or an I/O device.
Main Memory	Main Memory address space is the portion of the Memory address space that is allocated to the memory controller.
Memory Mapped I/O (MMIO)	Memory Mapped I/O address space is the portion of Memory address space that is allocated to I/O devices or zx1 mio registers.
I/O Port Space (IOS)	I/O Port space is a x86 legacy 16-bit address space that is allocated exclusively to I/O devices.

Table 3: Itanium-2 Address Spaces

1.4.4 Transaction Terminology

.

zx1 mio Name	Itanium-2	Definition
Main Memory Code Read	Memory Read	Requesting agent is reading a cache line containing code from main memory. No coherency snoop is performed.
Main Memory Read	Memory Read	Requesting agent is reading the most current copy of the cache line from main memory and does not need exclusive ownership. The requesting agent may mark the line exclusive if no other snooping agent owns the line.
Memory Read and Invalidate	Memory Read and Invalidate	Requesting agent is reading the most current copy of the cache line from main memory and needs to have exclusive access. All snooping agents must invalidate the line. (The requesting agent plans to modify a portion of the cache line.)
Memory Invalidate	Memory Read and Invalidate	Requesting agent needs exclusive access to a cache line from main memory and is invalidating the cache line in all snooping agents' caches. (The requesting agent plans to modify ALL of the cache line.)
Memory Read Current	Memory Read Current	Requesting agent (Always the IOC) is reading the most current copy of the cache line from main memory, but will never own the line. If the cache line is in another agent's cache, it does not need to change the cache line's state.
Main Memory Write or Explicit Writeback	Memory Write	Requesting agent is writing a modified cache line to main memory. No coherency snoop is performed.
Implicit Writeback	Implicit Writeback	Snooping agent is writing a modified cache line to an agent that requested the line earlier. The line may also be written to main memory by the memory controller.
DMA Read	Memory Read	An I/O device is reading the most current copy of the cache line from main memory and does not need exclusive ownership.
DMA Write	Memory Read and Invalidate & Memory Write	An I/O device is writing data to main memory. The I/O cache must first obtain exclusive ownership of the cache line before it modifies the cache line.
Internal Read	N/A	A special type of non-coherent DMA Read transaction. An I/O device is reading a cache line directly from the memory controller without issuing the read on the Itanium-2 bus. The read is non-coherent since it is not snooped on the Itanium-2 bus. Software must avoid conflicts when using Internal Reads.
Internal Read Return	N/A	A special way to return data for a DMA Read transaction. Main memory read return data is transferred directly from the memory controller to the I/O cache without using the Itanium-2 data bus.

zx1 mio Name	Itanium-2	Definition
Internal Writeback	N/A	A special type of Explicit Writeback for DMA Write transactions. The I/O cache transfers DMA Write data directly to the memory controller without using the Itanium-2 bus.
MMIO Read	Memory Read	Requesting agent (always a CPU) is reading from a non-coherent I/O device that is mapped into an MMIO region in memory address space.
MMIO Write	Memory Write	Requesting agent (always a CPU) is writing to a non-coherent I/O device that is mapped into an MMIO region in memory address space.
IO-Port Read	IO Read	Requesting agent (always a CPU) is reading from a non-coherent I/O device that is mapped into IO port space.
IO-Port Write	IO Write	Requesting agent (always a CPU) is writing to a non-coherent I/O device that is mapped into IO port space.
Purge TC	Purge TC	Requesting agent is invalidating an entry in the processor translation cache.
PCI Interrupt	Interrupt	An I/O device is sending an interrupt to a processor by asserting one of the PCI interrupt lines.
PCI Interrupt Transaction	Interrupt	An I/O device is sending an interrupt to a processor via a special case MMIO Write transaction to address 0xFEE0_0000 to 0xFEEF_FFFF.
Interrupt Acknowl-edge	Interrupt Acknowl-edge	Requesting agent is reading an interrupt vector from an 8259A or similar interrupt controller. Interrupt acknowledge transactions are routed to Rope 0.
Special Transaction	Special Transaction	Requesting agent is indicating some rare event to the system. Possible events are: NOP, Halt, Stop Grant Acknowledge and xTRP Update.

Table 4: Itanium-2 Transactions

1.4.5 Other Terminology

Itanium-2	Definition
Programmed I/O (PIO)	Programmed I/O transactions include MMIO Reads, MMIO Writes, IO-Port Reads, IO-Port Writes, Posted IO Reads and Posted IO Writes. All of these are initiated by a processor.
Peer-to-Peer (P2P)	Peer-to-Peer transactions include any access by an I/O device to MMIO address space or I/O Port space. The target may be a device on the same bus as the initiating I/O device or on a different bus. zx1 mio supports all types of Peer-to-Peer transactions when the originator and the target are on the same bus. zx1 mio only supports Peer-to-Peer MMIO Write transactions when the originator and target are on different buses.
Posted	Posted transactions are completed on the originating bus before being completed on the destination bus. This can only be done for write transactions. The address and data for the write transaction are “posted” in a queue and the transaction is completed later. This definition is borrowed from the PCI specification.

Table 5: Other Terminology

2 System Address Map

This chapter describes the physical address map and then presents the rationale behind the design of range registers in the zx1 mio and the rope guest (zx1 ioa) that determine how transactions are routed.

2.1 zx1 mio Physical Address Map

A version of the physical address map for Itanium-2 processors as implemented by the zx1 mio is shown in **Figure 3: zx1 mio Physical Address Map**.

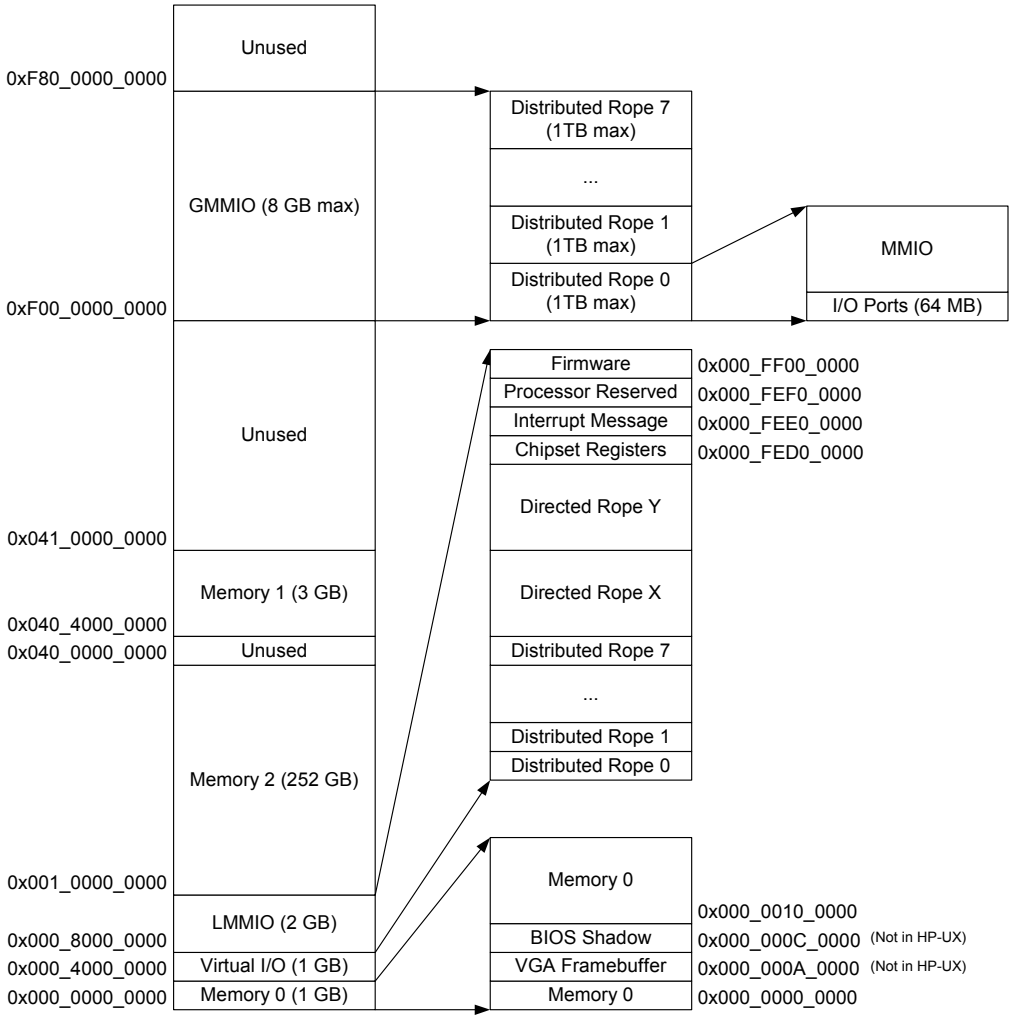

Figure 3: zx1 mio Physical Address Map

From the zx1 mio's perspective, there are several relevant regions:

The Compatibility Hole. 640K - 1M (0x000A_0000 – 0x000F_FFFF.) Named such because the region is required for compatibility with early PCs.

The I/O Virtual Region. The I/O virtual address space. This space is reserved for I/O devices that want to use the I/O PDIR to gain access to main memory. There is no memory mapped in this region. (If memory lived in this region, then 32-bit addressable I/O devices could not access it.) The memory that would have been mapped in this region is relocated to Memory 1.

The LMMIO Region. The I/O hole below 4GB. The upper 19MB of this region are reserved for the purposes stated in the address map. This region starts at 2G (0x8000_0000) and ends at 4GB-19MB-1 (0xFED0_0000). The memory that would have been mapped in this region is relocated to Memory 1.

The GMMIO Region. The I/O hole above 4GB. Both starting and ending addresses are programmable and will change from one system to the next. This region should be placed above all main memory. **NOTE: The GMMIO region is currently enabled in firmware, however, this region is not currently assigned to any cards/devices. The OSes are not yet using the GMMIO region.**

Firmware Space – Firmware space is 16MB located at 0xFF00_0000 – 0xFFFF_FFFF (at least for Itanium-2). Transactions to this space are forwarded to Dillon if it is present. Otherwise they are forwarded to rope 0. The zx1 mio supports cacheline sized reads to Dillon but a maximum of 16-byte reads if firmware is located below rope 0. The memory that would have been mapped in this region is relocated to Memory 1.

Memory Space. Memory space is divided into 3 separate regions (memory0, memory1 & memory2). Memory1 is unused until memory0 is full. Memory2 is unused until memory1 is full. Memory0 starts at address 0x0 and grows upward until it hits 0x3FFF_FFFF. Memory1 starts at 0x40_4000_0000 and grows upward to 0x40_FFFF_FFFF. Memory2 starts at address 0x01_0000_0000 and grows to the maximum supported memory capacity at address 0x3F_FFFF_FFFF.

2.2 The Compatibility Hole (640K to 1M)

The following table shows how the 640K to 1M area is used.

Address Range	Hexadecimal	Usage
640K - (768K-1)	A_0000 - B_FFFF	VGA frame buffer area
768K - (800K-1)	C_0000 - C_7FFF	VGA BIOS
800K - (896K-1)	C_8000 - D_FFFF	BIOS (ISA or PCI)
896K - (960K-1)	E_0000 - E_FFFF	Extended system BIOS
960K - (1M-1)	F_0000 - F_FFFF	System BIOS

Table 6: Compatibility Hole

Note that the VGA frame buffer is located in the first 128K. If a VGA device is present, transactions to that address range will be routed to the PCI bus that hosts the VGA device. The remaining 256K is all used for various forms of BIOS.

2.2.1 BIOS shadowing / write–protect

BIOS is firmware that is stored in a ROM and typically executed at power up. However, since ROM accesses are slow, it is desirable to copy the ROM content to RAM before executing it. In ISA systems, the ROM itself will be located in the area shown above, so the chip set has to jump through all kinds of hoops to allow execution of the code from RAM. That process is called BIOS shadowing.

Fortunately, systems using zx1 mio do not support ISA. In PCI, the BIOS ROMs are architected to be relocatable and their code need not be executed in place. Instead, the ROM itself is mapped to some region above 1 MB (presumably in the I/O hole below 4 GB) and the code is copied to the area below 1 MB. Unfortunately, some BIOSs are still written with the assumption that they execute out of a ROM and therefore, if they write to themselves, the content will not change. This means that writes to memory designated for BIOS in a zx1 mio system must be ignored, just like they would if the memory were an actual ROM. An exception must be made during the INIT process because PCI BIOSs do have an INIT function that is allowed to write to BIOS with the intent of shrinking it. Once the INIT function has been executed, the BIOS memory area must be "write–protected". What this means for hardware is that the area from 768K to 1M is actually claimed by memory, even though it is considered an I/O hole. The **zx1 mio's IOC does not respond to transactions in that range. The rope guest treats these transactions as normal DMA.** That memory is only used for BIOS. The bus interface block (BIB) in the zx1 mio implements the "write–protection" feature. If the bios_wp field of the BIB_MODE register is 0, then memory from 0xC_0000 to 0xF_FFFF can be read and written. If the bios_wp field of the BIB_MODE register is 1, then memory from 0xC_0000 to 0xF_FFFF can be read, but the writes will be discarded by the zx1 mio.

A note on terminology: BIOS shadowing really applies to ISA. The implementation described above is much simpler, because it is for PCI only. This implementation should not really be called BIOS shadowing. It is just BIOS write–protect.

2.2.2 VGA support

The zx1 mio will be designed to allow a VGA device to exist on any PCI or AGP bus. The PC architecture fundamentally only supports one VGA device in the system, so there is no support for multiple VGA devices, either on one PCI bus or on separate PCI busses. **If multiple VGA devices are installed, the firmware should disable all but one of them.** Of course, any number of non–VGA type graphics devices can be supported.

A VGA device claims the following address ranges:

- Memory space from 640K to 768K-1. (I.e., A_0000 to B_FFFF.)
- I/O port ranges: 0x3B0 through 0x3DF and their 10 bit aliases (if enabled). (Aliases are explained below.).

The enable bit of the VGA Routing Register (VGA_ROUTE) controls whether the IOC will claim VGA addresses or not. When the enable bit in VGA_ROUTE is 1, the IOC claims VGA addresses, regardless of what is programmed in the rest of its I/O space range registers. Otherwise, the IOC will not claim VGA addresses. In other words, for the VGA I/O port space, the VGA enable bit takes precedence over the I/O port distributed and directed range. The VGA_ROUTE also contains routing information that tells the IOC which rope contains the VGA device. If the IOC claims VGA transactions, it forwards them, according to the routing information, to the appropriate rope.

The rope guest (zx1 ioa) has two bits that control VGA routing. The following table summarizes the rope guest VGA routing bits.

zx1 ioa	Description
FV (forward VGA) bit in the Status, Information, and Control register	When this control bit is 1, the rope guest will allow local bus peer-to-peer to the VGA address space.
VPE (VGA peer-to-peer enable) bit in the Slave Control register	When this control bit is 1, the rope guest will allow remote bus peer-to-peer to the VGA address space.

Table 7: zx1 ioa VGA Routing Bits

The FV bit must be set if a VGA device is located on the zx1 ioa's PCI[X]/AGP bus. Only one zx1 ioa in the system should have its FV bit set. The following table outlines how the FV bit relates to the VPE bit in the zx1 ioa's SLAVE_CTRL register:

For Each zx1 ioa in the System	FV	VPE
VGA is present below this zx1 ioa	1	X
VGA is present in the system (not this zx1 ioa)	0	1
VGA is not present in the system	0	0

Table 8: zx1 ioa VGA Routing Possibilities

Note: The programming for VGA must be coordinated among the three registers containing VGA related bits (i.e., FV bit in the zx1 ioa's SIC register, VPE bit in the zx1 ioa's SLAVE_CTRL register, and finally the zx1 mio's VGA_ROUTE register).

2.2.3 ISA aliases

In the old days, ISA cards only decoded 10 bits of the I/O port address. With such cards, their I/O ports got aliased 64 times in the 16 bit I/O space. For example, on a 10 bit decoding VGA

ISA card, the I/O port at 0x3B0 is aliased to 0x7B0, 0xBB0, 0xFB0, 0x13B0, ..., and 0xFFB0. On PCI, of course, the VGA cards will decode all 16 bits. (Hopefully, all 32 bits.) But, they are allowed to map new registers at any of the aliases. PCI VGA cards are explicitly permitted to do this. So, when VGA Enable is set, the zx1 mio will claim all 64 aliases of the VGA I/O port space and forward them to PCI without modifying the address. This just means that the VGA I/O port decoding logic need not look at the upper 6 bits of the 16 bit I/O address. The zx1 mio provides a "VGA-lite" bit that can inhibit the VGA aliases from being forwarded to the VGA rope.

There is no special support for VGA palette snooping. (Palette snooping is best described in the PCI to PCI bridge architecture specification.) Devices that snoop VGA palette accesses must reside under the same zx1 mio PCI that the VGA device itself resides onThe I/O Hole Below 4 GB

The table below describes the address space for the I/O hole ending at 4 GB - 1.

Address Range	Hexadecimal	Usage
(2G) - (4G-19M-1)	8000_0000 - FECF_FFFF	LMMIO, **PCI memory mapped I/O**
(4G-19M) - (4G-18M-1)	FED0_0000 - FEDF_FFFF	Configuration space
(4G-18M) - (4G-17M-1)	FEE0_0000 - FEEF_FFFF	Interrupt message space
(4G-17M) - (4G-16M-1)	FEF0_0000 - FEFF_FFFF	Processor reserved
(4G-16M) - (4G-1)	FF00_0000 – FFFF_FFFF	Firmware space

Table 9: I/O Hole Below 4GB

LMMIO, or the PCI Memory mapped I/O range is the preferred range for allocating the memory mapped I/O space to PCI devices. This range is known as "LMMIO," for Less than 4 GB Memory Mapped I/O. Note that this range is used for all the different I/O buses in the system and must be divided among them. The IOC contains range registers that facilitate the division of this range in two different ways, distributed and directed. The distributed range is comprised of a lower and upper bound. The memory between the bounds is split up into 8 equal portions corresponding to each rope, hence the name "distributed range". The IOC allows for a maximum of two different directed ranges. Each of these ranges may be assigned, or "directed" to only one rope. These ranges are comprised of a lower and upper bound, as well as a rope designation.

Whenever zx1 mio systems are configured for multiple ropes per rope guest, the memory allocated to that rope guest's additional ropes by the IOC distributed range is also forwarded to the primary rope (e.g. if rope 0 and 1 are configured as a double rope then the address range allocated for rope 1 by the distributed range will be forwarded to rope 0). Space allocated by a distributed range can always be re-directed by assigning a directed range to the space (i.e. Directed ranges have higher priority than distributed ranges).

The rope guest also has range registers that tell it what portion of the address space is assigned to it. This information is used to determine if a peer-to-peer transaction is destined for another I/O device on the local I/O bus or if the peer-to-peer transaction needs to be forwarded to another rope.

Today's x86 systems have a default space where I/O APICs are mapped. The rope guest's I/O SAPIC (which is essentially equivalent to an I/O APIC) is **not** mapped in the default range from 0xFEC0_0000 – 0xFECF_FFFF. Rather, it is in the zx1 mio's configuration space. Future OSs are required to understand I/O APICs mapped to arbitrary addresses. The default I/O APIC range mentioned above is absorbed by LMMIO.

Configuration Space is where various registers in the zx1 mio are mapped. See the address map chapter for details on which range the zx1 mio claims. Note that the location of the rope guest configuration space is programmable through the ROPE_CONF_BASE register described in the IOC chapter. It is likely that ROPE_CONF_BASE will be located somewhere near the top of the LMMIO range and outside of both the distributed and directed portions of that range (the configuration space has priority over LMMIO directed and distributed ranges so overlap is possible). The rope guest configuration space will form its own distributed range (partitioned for 16 logical ropes) starting at ROPE_CONF_BASE and ending at ROPE_CONF_BASE + 128K, or 8K for each rope.

The Interrupt Message Space is used by PCI to send interrupt messages to processors. On the Itanium-2 bus, these messages are sent as special interrupt transactions, so this address range remains unused for the normal memory space. STOREs to that space by the processors will get converted to interrupt Special transactions for inter–processor interrupts. See the SAPIC spec for more information. The rope guest will always claim PCI writes to this space and convert them to interrupt transaction on the Itanium-2 bus. PCI reads of this space result in a target–abort. By default, the zx1 mio never responds to Itanium-2 bus transactions to this address range; they should never happen anyway. The zx1 mio's LMMIO range registers could be programmed to respond to that range, but it isn't recommended. Note: In today's PCs, the interrupt message space is used for the local APIC.

The Processor Reserved Range. From the zx1 mio's perspective, this range is unused.

The Firmware space, 16 MB of space from (4G-16M) to (4G-1), is used exclusively as processor dependent hardware space. The IOC claims this range and forwards the transactions to Dillon (if present) via the PDH bus. Dillon maps a boot ROM and various other resources into this space. Additional details are in the Dillon ERS. If Dillon is not present then these requests are forwarded to the PCI on rope 0. The zx1 mio supports cacheline sized requests to PDH if Dillon is present. If Dillon is not present then the requests must be 16 bytes or less.

2.3 MMIO above 4 GB

The zx1 mio allows additional memory mapped I/O above 4 GB. Such ranges are called "GMMIO," for Greater than 4 GB Memory Mapped I/O. If such a range is enabled, it should be placed above any main memory in the system. The IOC only allows for a distributed range in the GMMIO space. This distributed space can be divided among up among the logical ropes, as in the LMMIO space, but differs in that the first 64MB of each rope's space is optionally mapped to I/O port space. Processor writes to GMMIO space will be claimed by the zx1 mio and forwarded to the respective PCI bus. Note: each rope's GMMIO space must be a minimum of 4GB. The zx1 mio provides the ability to optionally re-map GMMIO into LMMIO on each rope. The zx1 mio does this translation by clearing the upper 32-bits of the address before forwarding the transaction to a rope.

2.4 I/O port space

The I/O port space is required purely for legacy reasons. x86 processors have always had it. There are many I/O devices out there that map their registers in I/O port space instead of memory space. Generally, new I/O devices should be designed to reside in memory space, but many standard PC functions have fixed allocations in the I/O port space and those are likely to continue to exist there.

There is really only 64 KB of useable I/O port space, so it is difficult to allocate. If a PCI device is designed so that it can be accessed through either memory space or I/O space, it should be mapped into memory space. I/O port space should be used only when there is no alternative.

Like the memory mapped I/O space, the I/O port space will also have to be distributed between different ropes. The entire 64KB I/O port space can be distributed equally between the logical ropes. I.e., each rope gets 8KB. The IOC supplies a programmable directed range which is intended to recover I/O port space lost due to unused ropes.

The zx1 mio has 3 ways to access PCI I/O port space: Itanium-2 I/O port space transactions, I/O port space in LMMIO ranges, and I/O port space in GMMIO ranges.

2.4.1 Itanium-2 I/O Port Space

When an IA-64 processor executes a load/store to an architected I/O space 64KB range, the processor emits an I/O port space transaction on the Itanium-2 bus. When the zx1 mio sees an Itanium-2 bus I/O port space transaction it claims the transaction if the zx1 mio's IOS_DIST_BASE register is enabled. The zx1 mio uses the following logic to find a destination rope for the transaction:

- If the I/O port address is a VGA one (0x3b0-0x3df inclusively) and VGA routing is enabled, then the VGA route register indicates the target rope.
- If the I/O port address is a 10 bit alias of a VGA register and VGA routing is enabled and VGA-lite is disabled, then the VGA route register indicates the target rope.
- If the I/O port address falls within the zx1 mio's I/O port directed range (and the directed range is enabled) then the EIOS_BASE register indicates the target rope.
- Otherwise bits [15:13] of the I/O port address are used to select a rope. This gives each rope (8 per zx1 mio) 8KB of I/O port space. Whenever zx1 mio systems are configured for multiple ropes per rope guest, the I/O port address allocated to that rope guest's additional rope is also forwarded to the primary rope (e.g. if rope 0 and 1 are configured as a double rope then the double rope will have 16KB of I/O port address allocated to it).

In all of the above cases writes to Itanium-2 I/O port space are non-posted. This space provides 64 KB of non-posted system I/O port space.

2.4.2 LMMIO I/O Port Space

If enabled, the zx1 mio claims a 1 MB densely packed range of LMMIO for I/O port space. This space is evenly distributed across each of the zx1 mio's 16 ropes. VGA routing is not done for access to this space. Bits [18:16] are used to select a target rope. Accesses to this space are non-

posted. This gives each rope 64 KB of non-posted I/O port space, duplicated twice (i.e. bit 19 is don't care).

In the 1MB above the LMMIO I/O Port Space are 8 areas of 128K each which allow access to the MMIO addresses 0xa0_0000 to 0xb_ffff for the VGA frame buffer area on each rope.

2.4.3 GMMIO I/O Port Space

If GMMIO is enabled, then each rope has a GMMIO range. The first 64 MB of each rope's GMMIO range is backed by 64 KB of I/O port space on that rope. The zx1 mio converts load/store accesses to the first 64 MB of each rope's GMMIO range to I/O port space transactions on the rope agent (if zx1 ioa then its PCI I/O port space transactions). Each 64MB I/O port space is broken into 16K spaces of 4KB. The 4KB pages map to exactly four bytes of I/O port space as illustrated below in Figure 4: GMMIO I/O Port Space Mapping. These mappings, taken together form a unique 64KB I/O port space for each of the 16 logical ropes. Itanium-2 transactions to these spaces will be claimed by the zx1 mio and then converted to I/O port space accesses to the corresponding 64KB of I/O port space. Accesses to these 64 MB GMMIO ranges are non-posted. This gives each rope 64 KB of non-posted I/O port space. With 16 ropes per zx1 mio this yields 1 MB of non-posted system I/O port space.

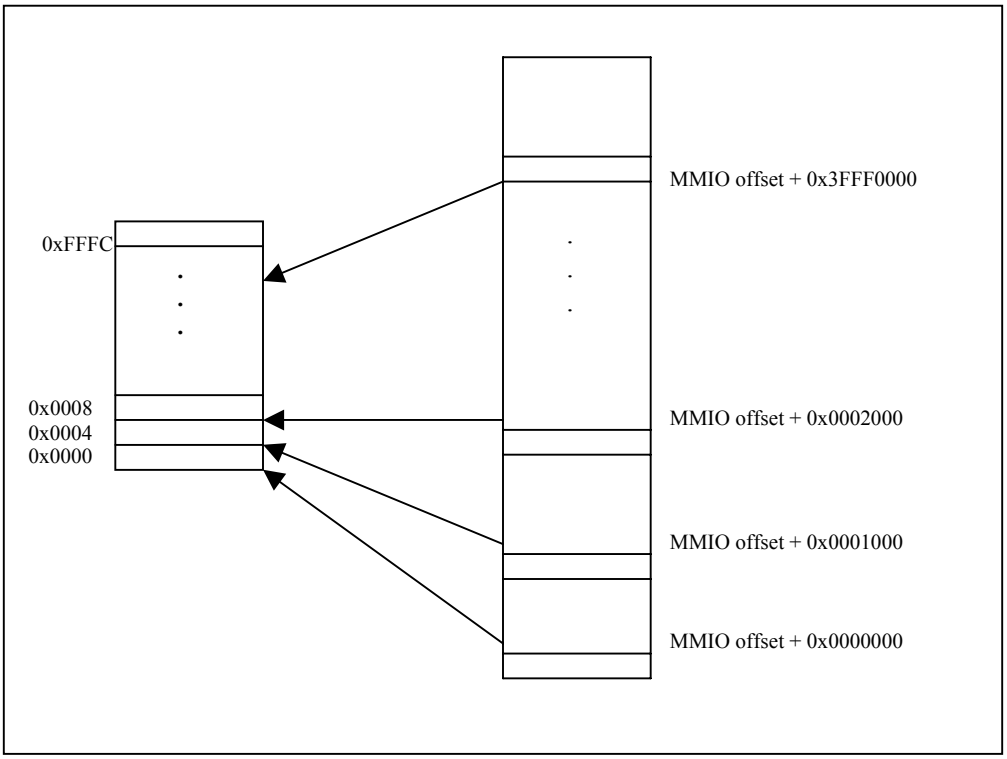

Figure 4: GMMIO I/O Port Space Mapping

2.4.4 Summary of I/O Port Spaces

Space	Space per rope	Space per system	Posted?	Directed I/O port range applies	VGA routing applies
Itanium-2 I/O Port Space	4 KB (less VGA aliases)	64 KB	Non posted	Yes	Yes
LMMIO I/O Port Space	64 KB	1 MB	Non posted	No	No
GMMIO I/O Port Space	64 KB	1 MB	Non posted	No	No

2.5 Range register programming

This section describes how the range registers in the zx1 mio fit together with range registers in the rope guest. The registers mentioned here are described in detail in the IOC chapter of this document and in the Slave Controller chapter of the rope guest ERS.

Some general rules:

- If two or four ropes are combined into a single logical rope then that rope gets the address space allocated to all of the ropes involved. (e.g. If rope0 is configured as a double-wide rope then it will get the addresses allocated to both rope0 and rope1).
- If a directed range overlaps the distributed range, the directed range takes precedence. Therefore, if a directed range is used to reassign the distributed space allocated to an unused rope, that address space is "recovered."
- It is illegal to have directed ranges overlap each other.

2.5.1 Range registers for the compatibility hole

The only zx1 mio and rope guest registers relevant to this range are the ones associated with VGA. These were explained in section 2.2.2.

2.5.2 Range registers for LMMIO space

First, distributed LMMIO in the IOC defines a range of memory to be divided equally among the up to 16 logical ropes. LMMIO_DIST_BASE and LMMIO_DIST_MASK specify the location and the size of the range. LMMIO_DIST_ROUTE specifies which address bits to use for selecting which rope gets a particular transaction. The rid_lsb field of the LMMIO_DIST_ROUTE register should be programmed with ($\log_2$ (size of the range in bytes / 8)). The divide by 8 in this formula comes from the fact that the range is being split across 8 ropes. Space allocated to unused ropes is wasted.

Each rope guest needs to know what portion of distributed LMMIO is assigned to it. This should be programmed in the rope guest's LMMIO_BASE and LMMIO_MASK registers. No routing information is needed. Note that only the particular $1/8^{th}$ of the distributed space allocated to this rope guest is programmed into these registers. Note that the LMMIO distributed range cannot

access above 0xfe00_0000, which means that for a 2GB LMMIO distributed area, rope 7 area is only 224MB instead of 256M like the rest.

To handle cases where a particular rope needs more space than is allocated by the distributed range, directed LMMIO ranges in the IOC should be use. Each directed LMMIO range is specified by three related registers: LMMIO_DIR_BASEx, LMMIO_DIR_MASKx, and LMMIO_DIR_ROUTEx. There are two such ranges, so x varies from 0 to 1. The Base register specifies the starting address of the range while the Mask register sets the size. Finally, the Route register specifies the rope to which addresses within this range will be routed. It should be apparent that ROUTE registers of directed ranges are different from ROUTE registers of distributed ranges.

If a directed range in the IOC is allocated to a particular rope, the rope guest on that rope needs to know that too. To allow this, an ELMMIO (Extra LMMIO) range is provided. It consists of a BASE and a MASK register. Since there is only one ELMMIO range, only one IOC directed range should be allocated to a rope – even though the IOC does not impose that restriction.

Finally, the rope guest needs to know about the size of the *system wide* LMMIO region. This is called WLMMIO and is programmed into WLMMIO_BASE and WLMMIO_MASK registers. PCI mastered transactions within WLMMIO will be treated as some form of peer-peer.

2.5.3 Range registers for GMMIO space

For GMMIO, the IOC supports only a distributed range. It works very much like the distributed LMMIO except that it works with addresses above 4GB. The rope guest has a corresponding GMMIO range that specifies how much GMMIO is allocated to that rope guest. Also, the rope guest has a WGMMIO range that specifies the size of the system wide GMMIO space. This is conceptually similar to WLMMIO.

2.5.4 Range registers for I/O port space

The IOC has an IOS_DIST_BASE register where only the enable bit is implemented. If that bit is a 0, the zx1 mio doesn't claim I/O port space transactions. The zx1 mio has an implied distributed range for I/O port space in the IOC. If IOS_DIST_BASE is enabled, the entire 64 KB is distributed across the 8 ropes. I.e., each rope gets 8 KB. Additionally, the IOC has one directed I/O port space range. This can be used to recover the space allocated to an unused rope.

The rope guest needs to know what portion of the I/O port space is allocated to it. The space supplied by the implied distributed range should be programmed in the IOS_BASE and IOS_MASK registers. If additional space is supplied with the directed range, EIOS_BASE and EIOS_MASK should be used to program that range as well. The rope guest does not need a "whole" IOS range because the entire 64 KB space is the whole IOS range. This is because main memory is never mapped to I/O port space.

I/O ports in the VGA range are always routed based on VGA related bits. I.e., VGA takes precedence over all other I/O range registers. Note that it is forbidden for a rope guest to do a P2P write to an address that turns into an I/O port space address (i.e. is deferred).

3 I/O Subsystem

The zx1 mio IOC design is centered around a small coherent cache connected via 8 ropes to chips that control industry standard I/O busses (zx1 ioa).

The following list outlines the feature set for the IOC.

- Support for 128-byte cachelines
- 3.2GB/s peak aggregate bandwidth
- Hardware enforced coherency
- 16-entry, fully associative coherent buffer for DMA writes (2KB)
- 16-entry re-order buffer for DMA reads (2KB)
- 16-entry, fully associative translation cache w/ programmable page sizes (I/O TLB)
- 8 high-speed rope connections (533MT/s)
- Support for single, double-wide, and quad-wide ropes
- Support for split transactions on all ropes.
- Parity protection on all data paths
- Internal path to main memory to minimize processor bus traffic
- Optional (via hints) enforcement of PIOW/DMAR ordering constraints
- Supports inter-rope P2P writes.

3.1.1 Address Decoding

The targets for PIO transactions that are decoded by the PIO control block are listed in Table 10. Note that all address ranges must be power of 2 size and naturally aligned. All addresses in the table are assumed to be 44-bit addresses.

Destination	Starting Address	Block Size	Description
zx1 mio configuration	FED0_0000	64KB	All zx1 mio registers will be accessed through this space. All registers are 8 bytes. This space allows for 16 functions at 4K bytes apiece.

Destination	Starting Address	Block Size	Description
Rope guest configuration	programmable (8000_0000 – FFFE_0000)	128KB	The base address for this range is specified in the ECNFG_BASE register. There are 2 functions (8KB) allocated to each rope guest. The upper 64KB is reserved for future expansion.
Reserved	FED0_4000	16KB	Reserved for future expansion. Note that this Reserved area is right in the middle of the 64KB zx1 mio Configuration area.
Firmware Space	FF00_0000	16MB	All accesses in this range will be forwarded to the PDH bus if Dillon is present. Otherwise they will be sent to rope0. This range has lower priority than any of the other I/O ranges (e.g. If there is overlap between an LMMIO directed range and the PDH range, the LMMIO range will be used).
PCI I/O port space distributed range	0	64KB	I/O port space transactions will be divided into 8 equal segments. Addresses in each segment will be forwarded to the corresponding rope and result in an I/O space transaction on PCI.
PCI I/O port space directed range	programmable (0 – F000)	4KB-64KB	I/O port space transactions in this address range will be forwarded to a specific rope and result in an I/O space transaction on PCI.
PCI LMMIO space distributed range	programmable (8000_0000 – FFF0_0000)	1M-2GB	Addresses in this space will be divided into 8 or 16 equal segments (one per rope). Transactions will be forwarded to the appropriate rope and result in a PCI memory transaction with a 32-bit address.
PCI LMMIO space directed range (x2)	programmable (8000_0000 – FFF0_0000	1M-2GB	Transactions in this address space will be forwarded to the appropriate rope and result in a PCI memory transaction with a 32-bit address. The destination rope is programmable. There are two sets of PCI LMMIO ranges.
PCI GMMIO distributed memory space	programmable (1_0000_0000 – FFF_0000_0000)	4G–8TB	Addresses in this space will be divided into 16 equal segments. The first 64MB of each segment will optionally be mapped down to 64KB of posted IOP for that rope. All other addresses will result in a PCI memory transaction with a dual address cycle (DAC). Bits 43:0 of the Itanium-2 address will be used to generate the PCI address for these transactions. This range supports a mode that forces bits 63:32 of the PCI address to

Destination	Starting Address	Block Size	Description
			0's and therefore avoids the DAC on PCI.
VGA MMIO space	A_0000	128K	Required for compatibility with OS's that depend on VGA. Addresses in this range will be forwarded to the specified rope. Note that this space can be disabled if VGA is not required.
VGA I/O port space	3B0	48B + Aliases	Required for compatibility with OS's that depend on VGA. Addresses in this range will be forwarded to the specified rope. The Aliases imply that only the 10 least significant bits of the 16-bit IOS address should be decoded. Note that this space can be disabled if VGA is not required. This range supports a mode that will not include the aliases.

Table 10: PIO Targets Decoded by the IOCC

Note that the zx1 mio will always forward an interrupt acknowledge transaction from the Itanium-2 bus to rope 0.

3.2 Function 0 Registers

This section gives bit level descriptions of all the registers residing in function 0 of the zx1 mio
Function 0 ID Register

The Function ID Register is used to uniquely identify the function.

MSB	Function 0 ID Register (0xFED0_0000)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reseved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
function_id[15:0]																vendor_id[15:0]															
Power On Initialization																															
16'h1229																16'h103C															

Size: 64-bit only		
Field	**Access**	**Description**
vendor_id[15:0]	R	Always set to 16'h103c
function_id[15:0]	R	Always set to 16'h1229

Register 1: Function 0 ID Register

3.2.1 Function 0 Class (FC) Register

The Function Class (FC) register defines Function 0's class code, revision level and line size. It is read-only.

MSB	Function 0 Class Register (0xFED0_0008)	LSB

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reseved																								line_size[7:0]							
Power On Initialization																															
24'h000000																								8'h20							

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
class_code[23:0]																								revision[7:0]							
Power On Initialization																															
24'h068000																								8'h23							

Size: 64-bit only		
Field	**Access**	**Description**
revision[7:0]	R	Will start out at 8'd0 for the first version of the chip and will be incremented in future revisions if there are any changes to this function. A 8'h23 for TR2.3, 8'h22 for TR2.2, 8'h20 for TR2.0, TR2.1.
class_code[23:0]	R	Always set to 24'h068000 (other bridge)
line_size[7:0]	R	Always set to 8'h20 to reflect a 128-byte cacheline

Register 2: Function Class (FC) Register (function 0)

3.2.2 Module Info (MI) Register

The Module Info (MI) register in the zx1 mio is present only in function 0. It is used to identify the module and indicate which functions are present in the module.

MSB	Module Info Register (0xFED0_0100)																															LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reseved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
functions_present[15:0]																module_id[15:0]															
Power On Initialization																															
16'h0703																16'h000A															

Size: 64-bit only		
Field	**Access**	**Description**
module_id[15:0]	R	Always set to 16'h000A
function_present[15:0]	R	Always set to 16'h0703. This indicates that zx1 mio uses functions 0, 1, 8, 9, & 10.

Register 3: Module Info (MI) Register

3.2.3 Address Range Registers

The IOCC contains a set of programmable read/write address range registers. The range registers set bounds for the lower (less than 4 GB, i.e. LMMIO) and upper (greater than 4 GB, i.e. GMMIO) Memory-Mapped I/O holes, and I/O Port Space ranges. The bit mappings of the range registers are documented in this chapter. The zx1 mio has 2 directed ranges and one distributed range in LMMIO space. It has a single distributed range in GMMIO space. The intent is that almost all devices map through the LMMIO distributed space. Large PIO devices can either use GMMIO, or the LMMIO distributed area can be configured to give each rope (except for 7) 256MB (512MB for double ropes). Rope 7 will have 224MB instead of 256MB.

If range registers are mapped to conflict with each other (at least LMMIO distributed is expected to do this), the following priorities detail which range will be selected (high to low priority).

1) zx1 mio registers.

2) Rope guest registers.

4) lmmio0 directed range

5) lmmio1 directed range

7) GMMIO area.

8) PDH area.

9) LMMIO distributed.

For I/O space access, the priority is VGA I/O, IOS directed, then IOS distributed.

3.2.3.1 LMMIO Directed Range Registers

The zx1 mio supports 2 directed LMMIO address ranges. Each directed LMMIO address range can be assigned to any of the ropes connected to the zx1 mio, and multiple ranges can be assigned to a single rope. There are three registers associated with each of the LMMIO directed ranges which set the base address, size of the range, and the target rope.

3.2.3.1.1 LMMIO Directed Base Register

The LMMIO_DIR_BASE Register is instantiated two times, once for each LMMIO directed range. It defines a 1 MB-aligned base address for the directed LMMIO range assigned to the given rope, and enables decoding to that range. The base of the LMMIO range must be greater than or equal to 2 GB and less than 4 GB (i.e. LMMIO ranges match only 32-bit addresses).

MSB	LMMIO_DIR_BASE[0],[1] Register ([0xFED0_0300],[0xFED0_0318])																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	BASE_ADDR											Reserved																			RE
Power On Initialization																															
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
RE	R/W	0	0	Range Enable – This bit must be asserted for any address to ever hit this range.
BASE_ADDR	R/W	0	30:20	Specifies the starting address of the range. Bit 31 of the address is assumed to be a 1 and all other bits of the address are assumed to be 0.

Register 4: LMMIO_DIR_BASE(N)

3.2.3.1.2 LMMIO Directed Mask Register

The LMMIO_DIR_MASK Register is instantiated two times, once for each directed LMMIO range. It defines the address mask that defines the size of that range. The size can be any power of 2 from 1MB to 2 GB. When a bit is asserted in the mask register, that bit position in the address will be compared to that bit position in the base address. If there is a 0 in the mask register, then that bit position will be ignored when doing the decoding. If the address matches the base register in all bit positions that are being compared (as specified in the mask register) then the address is considered to fall into the range.

MSB	LMMIO_DIR_MASK[0],[1] Register ([0xFED0_0308],[0xFED0_0320])																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	MASK											Reserved																			
Power On Initialization																															
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
MASK	R/W	0	30:20	Specifies the bit positions in the incoming address that need to be compared with the contents of the base register to determine if the address fall in the specified range.

Register 5: LMMIO_DIR_MASK

3.2.3.1.3 LMMIO Directed Range Route Register

The LMMIO_DIR_ROUTE register is instantiated two times, once for each directed LMMIO range. It is used to route a transaction hitting the given LMMIO range to a rope. Note that up to 16 ropes can be specified.

MSB	LMMIO_DIR_ROUTE[0],[1] Register ([0xFED0_0310],[0xFED0_0328])																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																													ROUTE		
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
ROUTE	R/W	0	2:0	Specifies the target rope for any address that hits in the respective directed range.

Register 6: LMMIO_DIR_ROUTE

3.2.3.2 LMMIO Distributed Range Registers

The zx1 mio supports a single distributed LMMIO address range. Address space in this range is divided into 8 equal regions which are assigned to each of zx1 mio's 8 ropes.

The LMMIO_DIST_BASE and LMMIO_DIST_MASK registers define the base and mask for the LMMIO distributed range, respectively. Their bit mappings are the same as those of the LMMIO_DIR_BASE and LMMIO_DIR_MASK registers, respectively. The LMMIO_DIST_ROUTE Register is used to route an address falling within the LMMIO distributed range to one of 8 ropes.

NOTE: The top 32MB of the LMMIO distributed area are disabled if the top of LMMIO distributed space is at (4GB-1). That is, all addresses at or above 0xfe00_0000 cannot be accessed via the LMMIO distributed region. This was done to allow LMMIO distributed range to be programmed with a 2GB size, and not have rope 7 interfere with HPA space or various Intel reserved address areas.

3.2.3.2.1 LMMIO Distributed Base Register

The LMMIO_DIST_BASE defines a 1 MB-aligned base address for the distributed LMMIO range and enables decoding to that range. The base of the LMMIO range must be greater than or equal to 2 GB and less than 4 GB (i.e. LMMIO ranges match only 32-bit addresses). The zx1 mio cannot access any address greater than 0xfe00_0000 with the LMMIO distributed region.

MSB	LMMIO_DIST _BASE Register (0xFED0_0360)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	BASE_ADDR											Reserved																			RE
Power On Initialization																															
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
RE	R/W	0	0	Range Enable – This bit must be asserted for any address to ever hit this range.
BASE_ADDR	R/W	0	30:20	Specifies the starting address of the range. Bit 31 of the address is assumed to be a 1 and all other bits of the address are assumed to be 0.

Register 7: LMMIO_DIST_BASE

3.2.3.2.2 LMMIO Distributed Mask Register

The LMMIO_DIST_MASK register defines the address mask that defines the size of the range. The size can be any power of 2 from 1MB to 2GB. When a bit is asserted in the mask register, that bit position in the address will be compared to that bit position in the base address. If there is a 0 in the mask register, then that bit position will be ignored when doing the decoding. If the address matches the base register in all bit positions that are being compared (as specified in the mask register) then the address is considered to fall into the range. Addresses above 0xfe00_0000 cannot be accessed using LMMIO distributed area.

MSB	LMMIO_DIST_MASK Register (0xFED0_0368)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	MASK											Reserved																			
Power On Initialization																															
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
MASK	R/W	0	30:20	Specifies the bit positions in the incoming address that need to be compared with the contents of the base register to determine if the address fall in the specified range.

Register 8: LMMIO_DIST_MASK

3.2.3.2.3 LMMIO Distributed Range Route Register

The LMMIO_DIST_ROUTE register is used to specify the bits of the address that contain the target rope number for addresses that hit in the LMMIO Distributed range.

SB	LMMIO_DIST_ROUTE Register (0xFED0_0370)																															LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
ROUTE						Reserved																										
Power On Initialization																																
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
ROUTE	R/W	0	63:58	Specifies the bit position in the address that represents the least significant bit of the 3-bit target rope number. The legal values for zx1 mio are 16 – 28. Set to top masked bit –2.

Register 9: LMMIO_DIR_ROUTE

3.2.3.3 GMMIO Distributed Range Registers

The zx1 mio decodes a single GMMIO (>4GB) address space that is divided into equal sized chunks and distributed to multiple ropes.

You should *not* program GMMIO to claim addresses below 4GB (i.e. the GMMIO_BASE register is the initial value of all 0's) or you will suffer a quick death.

You should also *not* overlap the GMMIO space with any other space.
Please refer to a system memory map for the recommended GMMIO address location.

3.2.3.3.1 GMMIO Distributed Range Base Register

MSB	GMMIO_DIST _BASE Register (0xFED0_0378)																													LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																				BASE_ADDR											
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																													RL	PD	RE
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
RE	R/W	0	0	*Range Enable* – This bit must be asserted for any address to ever hit this range.
PD	R/W	0	1	*I/O Port Space Disable* – When asserted, the lower 64MB of each segment of the distributed range is treated just like the rest of the segment (MMIO).
RL	R/W	0	2	*Re-map to LMMIO* – When asserted, the upper 32-bits of the address will be cleared before sending the transaction to the rope.
BASE_ADDR	R/W	0	43:32	Specifies the starting address of the range.

Register 10: GMMIO_DIST_BASE Register

The GMMIO_DIST_BASE Register defines the base address used for the distributed range above 4 GB. GMMIO ranges match only greater than 32-bit addresses.

Bit 0 is the range enable. If it is 1, decoding of addresses within the distributed GMMIO range are decoded, and transactions hitting the range are forwarded to the appropriate rope. (as determined by the GMMIO Mask and GMMIO ROUTE registers, defined below). If the bit is 0, transactions are not forwarded.

When the PD bit is not set, a transaction falling within the lowest 64 MB of the GMMIO segment assigned to a PCI bus (i.e. the range addr_base to addr_base+64MB) is converted to an I/O Port Space transaction for that PCI bus. More detail on the addressing scheme used for I/O Port Space is given in the Operation Overview chapter. When the PD bit is asserted these transactions in the lower 64MB will be treated like the rest of the range.

The RL bit is asserted to cause transactions in this range to be remapped below 4G. This is accomplished by simply clearing all bits past bit 31. If none of the address bits 43-32 are set when GMMIO is enabled, the system will die because memory addresses will be claimed as I/O.

3.2.3.3.2 GMMIO Distributed Range Mask Register

The GMMIO_DIST_MASK register defines the address mask that defines the size of the range. The size can be any power of 2 from 4GB to 16 TB. When a bit is asserted in the mask register, that bit position in the address will be compared to that bit position in the base address. If there is a 0 in the mask register, then that bit position will be ignored when doing the decoding. All address bits above bit 43 will always be compared and all bits below bit 32 will never be compared. If the address matches the base register in all bit positions that are being compared (as specified in the mask register) then the address is considered to fall into the range. When the address falls in this range it will be forwarded to the rope specified in the GMMIO_DIST_ROUTE register.

MSB	GMMIO_DIST_MASK Register (0xFED0_0380)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																				MASK											
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
MASK	R/W	0	43:32	Specifies the bit positions in the incoming address that need to be compared with the contents of the base register to determine if the address fall in the specified range.

Register 11: GMMIO_DIST_MASK

3.2.3.3.3 GMMIO Distributed Range Route Register

The GMMIO_DIST_ROUTE register is used to specify the bits of the address that contain the target rope number for addresses that hit in the GMMIO Distributed range. Note that up to 16 ropes can be specified. .

MSB	GMMIO_DIST_ROUTE Register (0xFED0_0388)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
ROUTE						Reserved																									
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
ROUTE	R/W	0	63:58	Specifies the bit position in the address that represents the least significant bit of the 3-bit target rope number. This has the effect of dividing the range into equal chucks. The legal values for zx1 mio are 28-41.

Register 12: GMMIO_DIST_ROUTE Register

3.2.3.4 IOS Distributed Range

The IOS Distributed Range is used for routing I/O port space transactions on the Itanium-2 bus to the correct rope. The IOS distributed range is 64KB. In routing an I/O Port Space transaction to a rope, VGA range decode take precedence over distributed decode. In other words, the decode algorithm first checks if there is a match against the VGA I/O Port Space range and if it doesn't match, the transaction is routed to the rope whose ID is given by the distributed range. Note that the range assumes a 32-bit address but only 16-bits of that address will ever be forwarded to a rope (i.e. bits 31:16 will always be cleared before forwarding the address to a rope). In an IA64 processor, bits 31:16 will be cleared anyway according to the Itanium-2 spec.

3.2.3.4.1 IOS Distributed Base Register

The IOS_DIST_BASE is used only for the purpose of enabling the range. It is assumed that the base address of the range is always located at 0x0.

MSB	IOS_DIST _BASE Register (0xFED0_0390)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																															RE
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
RE	R/W	0	0	Range Enable – This bit must be asserted for any address to ever hit this range.

Register 13: IOS_DIST_BASE

3.2.3.4.2 IOS Distributed Range Mask Register

The IOS distributed range mask register is used to specify the size of the range. The range is 64KB. This register should be left at its default value, but writing to it will do absolutely nothing except change what value is read from this register.

MSB	IOS_DIST_MASK Register (0xFED0_0398)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
12'hFFF												RESERVED				Reserved															
Power On Initialization																															
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
RESERVED	R/W	4'hF	19:16	Bits should be left high.

Register 14: IOS_DIST_MASK

3.2.3.4.3 IOS Distributed Range Route Register

The IOS_DIST_ROUTE register is used to specify the bits of the address that contain the target rope number for addresses that hit in the IOS Distributed range. The range can be divided into either 8 or 16 ropes.

MSB	IOS_DIST_ROUTE Register (0xFED0_03A0)																															LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
ROUTE						Reserved																										
Power On Initialization																																
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																																
Power On Initialization																																
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
ROUTE	R/W	6'd12	63:58	Specifies the bit position in the address that represents the least significant bit of the 3-bit target rope number. The legal value for zx1 mio is 13. This gives 8 8KB regions.

Register 15: IOS_DIST_ROUTE

3.2.3.5 IOS Directed Range Registers

The zx1 mio has a single IOS directed range. It takes precedence over the IOS distributed range. It is provided primarily for the case where a portion of the IOS distributed space needs to be re-directed to a different rope.

3.2.3.5.1 IOS Directed Base Register

The IOS_DIR_BASE Register defines a 256 byte-aligned base address for the directed I/O Port Space range. This register also contains the range-enable bit used to enable the range.

MSB	IOS_DIR_BASE Register (0xFED0_03C0)																															LSB
63	62	61	60	59	8	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	4	23	22	1	20	19	18	17	16	15	14	13	2	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																BASE_ADDR								Reserved							RE
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
RE	R/W	0	0	Range Enable – This bit must be asserted for any address to ever hit this range.
BASE_ADDR	R/W	0	15:8	Specifies the starting address of the range.

Register 16:IOS_DIR_BASE

3.2.3.5.2 IOS Directed Mask Register

The IOS_DIR_MASK Register defines the address mask that determines the size of the range. The size can be any power of 2 from 256 bytes to 64KB. When a bit is asserted in the mask register, that bit position in the address will be compared to that bit position in the base address. If there is a 0 in the mask register, then that bit position will be ignored when doing the decoding. If the address matches the base register in all bit positions that are being compared (as specified in the mask register) then the address is considered to fall into the range. It is assumed that IOS transactions have a 32-bit address.

MSB	IOS_DIR_MASK Register (0xFED0_03C8)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16'hFFFF																MASK								Reserved							
Power On Initialization																															
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
MASK	R/W	0	15:8	Specifies the bit positions in the incoming address that need to be compared with the contents of the base register to determine if the address fall in the specified range.

Register 17: IOS_DIR_MASK

3.2.3.5.3 IOS Directed Range Route Register

The IOS_DIR_ROUTE register is used to route a transaction hitting the directed IOS range to a specific rope.

MSB	IOS_DIR_ROUTE Register (0xFED0_03D0)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																													ROUTE		
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
ROUTE	R/W	0	2:0	Specifies the target rope for any address that hits in the respective directed range.

Register 18: IOS_DIR_ROUTE

3.2.3.6 Rope Configuration Base Register

The ROPE_CONFIG_BASE Register is used to set the base address for the LMMIO distributed range assigned to rope guest Configuration Space. This range is aligned on a 128 KB boundary and divided into 16 chunks of 8 KB each. Since the range size is hard-wired, there is no need for mask or route registers. The upper 64KB is reserved. This range takes precedence over the LMMIO distributed or directed ranges.

MSB	ROPE_CONFIG _BASE Register (0xFED0_03A8)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reserved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	BASE_ADDR														Reserved																RE
Power On Initialization																															
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
RE	R/W	0	0	Range Enable – This bit must be asserted for any address to ever hit this range.
BASE_ADDR	R/W	0	30:17	Specifies the starting address of the range. Bit 31 of the address is assumed to be a 1 and all other bits of the address are assumed to be 0.

Register 19: ROPE_CONFIG_BASE

3.2.3.7 VGA Range Routing Register

The VGA Range Routing Register defines how MMIO and I/O Port Space transactions falling within the hard-wired VGA address range are routed to PCI buses. This register implements a range enable and 4 bits of route to identify the rope attached to a PCI bus with VGA devices. The VGA range takes precedence over all other address ranges. The VL bit is used to disable the decoding of the VGA aliases (see the system map chapter for more details)

SB	VGA_ROUTE Register (0xFED0_03B0)																															LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
RE	VL	Reserved																													
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																												ROUTE			
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only				
Symbol	**Access**	**Reset**	**Bit Position**	**Name**
RE	R/W	0	63	*Range Enable* – Must be asserted for any address to hit this range. This bit is only set if there is VGA graphics in the system.
VL	R/W	0	62	*VGA-lite* – When asserted, the VGA aliases are not considered part of this range.
ROUTE	R/W	0	3:0	Specifies the target rope for any address that hits in the respective directed range. Please leave bit 3 equal to 0.

Register 20: VGA_ROUTE

3.3 Function 1 Registers (IOC)

Note that these registers may be written to using any byte enables permitted for an 8 byte write. That is, these registers pay attention to the byte enable bits on an 8 byte Itanium-2 write.

3.3.1 Function 1 ID Register

The Function ID Register is used to uniquely identify the function.

MSB	Function 1 ID Register (0xFED0_1000)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reseved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
function_id[15:0]																vendor_id[15:0]															
Power On Initialization																															
16'h122a																16'h103C															

Size: 64-bit only		
Field	**Access**	**Description**
vendor_id[15:0]	R	Always set to 16'h103c
function_id[15:0]	R	Always set to 16'h122a

Register 21: Function 1 ID Register

3.3.2 Function 1 Class (FC) Register

The Function Class (FC) register defines Function 1's class code, revision level and line size. It is read-only.

MSB	Function 1 Class Register (0xFED0_1008)	LSB

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reseved																								line_size[7:0]							
Power On Initialization																															
24'h000000																								8'h20							

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
class_code[23:0]																								revision[7:0]							
Power On Initialization																															
24'h068000																								8'h23							

Size: 64-bit only		
Field	**Access**	**Description**
revision[7:0]	R	Will start out at 8'd0 for the first version of the chip and will be changed in future revisions if there are any changes to this function. A 8'h23 for TR2.3, 8'h22 for TR2.2.
class_code[23:0]	R	Always set to 24'h068000 (other bridge)
line_size[7:0]	R	Always set to 8'h20 to reflect a 128-byte cacheline

Register 22: Function Class (FC) Register (function 1)

3.3.3 Rope Configuration Register

The rope configuration register contains the information that enables or disables each of the rope controllers as well as enabling two and four wide rope communication. Writes of the register only take affect after a rope soft reset to the affected rope(s) is initiated by a write to the LBA_Port(N)_CTRL register. .

MSB	ROPE_CONFIG Register (0xFED0_1040)	LSB

63	62	61	6	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reseved																															
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																U	U	Q4	Q0	D6	D4	D2	D0	U	U	U	U	U	U	U	U
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Size: 64-bit only		
Field	**Access**	**Description**
D0	R/W	Configures rope 0 to operate as double-wide (ignored if Q0 == 1)
D2	R/W	Configures rope 2 to operate as double-wide (ignored if Q0 == 1)
D4	R/W	Configures rope 4 to operate as double-wide (ignored if Q1 = 1)
D6	R/W	Configures rope 6 to operate as double-wide (ignored if Q1 = 1)
Q0	R/W	Configures rope0 to operate as quad-wide
Q4	R/W	Configures rope4 to operate as quad-wide
U	R/W	Unused – bits are R/W for backwards compatibility but the contents of these bits are ignored by the hardware.

Register 23: Rope Configuration Register

At reset, both the zx1 mio and the rope slave (zx1 ioa) are configured for single wide rope communication. To change to two wide the following steps must be taken:

1 Read the rope guest's rope configuration register (offset 0x0610) to determine if the rope is connected for two wide or 4 wide rope communication.

2 Write a 1 to the rf bit of the corresponding rope control register. This will reset the selected zx1 mio rope controller and the rope guest below it.

3 Read the rope control register until the rc bit (bit 32) is a 0.

4 Write to the rope guest's rope configuration register, setting it to the appropriate width. Communication with the rope guest is not possible until the next two steps are executed.

5 Write to the zx1 mio's rope configuration register, setting it to the same width as was programmed in the rope guest.

6 Write to the rf bit of the corresponding rope control register. This will reset the selected zx1 mio rope controller and the rope guest below it.

7 Read the rope control register until the rc bit (bit 32) is a 0.

8 Start a software timer to count to 1ms. This is the minimum PCI bus reset time.

9 Wait for the 1ms timer that was started in step 8 above.

10 Write a 0 to the rf bit of the rope guest Status_Control register. This brings the I/O bus out of reset. Communication with PCI devices is now possible.

3.3.4 LBA_Port(N)_CNTRL Register

LBA_Port(N)_CNTRL (0<=N<=7) registers provide the ability to perform a soft reset to a specific rope. They are also used to clear all the error registers for a specific rope controller. In addition, this register specifies whether PIO transactions are completed with a –1 or a hardfail when the rope controller is in fatal mode.

MSB	LBA_Port(N)_CNTRL Register (0xFED0_1200 – 0xFED0_1238)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
Reseved																															RC
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																									HF	CE	CL	Reseved			RF
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only		
Field	**Access**	**Description**
RF	W	*Reset Function* - Writing a 1 will force a soft reset the corresponding LBA_Port
RC	R	*Reset Complete* - Will read 1 while a LBA_Port is being reset. Will go to 0 on its own once reset is complete.
HF	R/W	*Hard Fail* - If set, returns HardFails instead of all 1s on reads with errors.
CE	R/W	*Clear enable* – Enables software to clear the error log by writing a 1 to the CL bit. Cleared by hardware the contents of the error log registers change.
CL	R/W	*Clear Error Log* - Clears the associated Error registers if CE is currently set. Will be cleared by hardware if the attempt to clear the error logs is unsuccessful.

Register 24: LBA_Port(N)_CNTRL

The behavior of the CE/CL bits when clearing error registers is clarified in Table 11:

Value Written		Previous Value		Final Value		Result
CL	CE	CL	CE	CL	CE	
0	0	X	X	0	0	Unchanged or new error
0	1	X	X	0	0	New Error Logged
0	1	X	X	0	1	Logs Unchanged
1	0	X	0	0	0	Unchanged or new error
1	0	X	1	0	0	Not Cleared and New Error
1	0	X	1	1	0	Cleared
1	1	X	X	U	U	Undefined

Table 11: CL/CE Bit Behavior

The procedure for clearing the error log is:

1. Write to the LBA_Port(N)_CNTRL register and assert the CE bit.
2. Read the error log.
3. Write to the LBA_Port(N)_CNTRL register and assert the CL bit.

4. Read the LBA_Port(N)_CNTRL register. If the CL bit is set then the error log was cleared. Otherwise, the error log has been modified since it was read so it was not cleared and the procedure must be repeated.

NOTE: The logic for an LBA Port (i.e., a rope controller) doesn't clear an error internally until the *log* has been cleared. Thus, any inbound rope activity which occurs *before* the log has been cleared will be treated as if the error were still in effect.

3.4 Transaction Overview

This section describes the basic transaction types supported by the zx1 mio (Memory accesses, PIO, DMA, peer to peer, interrupt message, interrupt acknowledge transactions, etc). Some knowledge of PCI protocol is helpful when reading this section.

3.4.1 PIO Transactions

PIO Transactions are read and write transactions that are initiated by a CPU with a non-memory target. These transactions can vary in size from 1-byte to 1 cacheline. Possible targets include PCI memory space, PCI I/O space, zx1 mio registers, etc. The following sections give examples of each supported type of PIO transaction. A peer-to-peer (P2P) transaction is a special case of a PIO transaction in that is initiated by a PCI device.

3.4.1.1 MMIO Write Transaction

Any time a memory write transaction appears on the Itanium-2 bus, the IOCC will compare it against the I/O addresses allocated to the zx1 mio. If the address is an I/O address, the IOCC determines the target for the transactions (rope, register or PDH) and forwards the address, transaction type, and rope number to appropriate target. The BIB responds with a normal no data response during the response phase on the Itanium-2 bus. During the data phase of the transaction, the BIB will tell the IOCC that the data is I/O data and the IOCC will grab the data directly from the Itanium-2 bus pads. The IOCC will convert the ECC to byte parity and combine it with the address and forward it to the correct rope controller. The RQC sends the address, transaction information, and data over the rope to the rope guest. The rope guest obtains ownership of the I/O bus and masters a memory write transaction on the I/O bus. If the I/O device performs a retry or disconnect during the data phase, the rope guest will retry or resume the transaction later. Flow control at the Itanium-2 bus interface and ropes controller is performed as described in the chapters for those blocks.

Memory write transactions to sequential addresses can be coalesced. This is called write combining in Itanium-2 bus terminology. The Itanium-2 processor is capable of doing write combining in the CPU for a burst size of 1-8, 16, 32 or 128 bytes. Eventually, the rope guest will see a single PCI write address and multiple data values so that the rope guest can attempt to burst it out to the PCI device without the overhead associated with additional arbitration and addressing phases.

3.4.1.1.1 Register writes

Writes to the zx1 mio's and the rope guests' registers are a special case of MMIO writes. These writes are either for registers inside the zx1 mio or for registers inside one of the rope guest devices. Writes destined for the zx1 mio registers are decoded by the IOCC and issued on the zx1 mio's internal regbus. Writes destined for the rope guest devices are decoded by the IOCC and forwarded to the appropriate rope.

3.4.1.2 MMIO Read

The Itanium-2 processor can initiate uncacheable memory reads of 1-8 or 16 bytes. Anytime a memory read transaction appears on the Itanium-2 bus, the zx1 mio's IOCC will compare the address against the MMIO address ranges allocated to the zx1 mio. If the address falls in any of these ranges the IOCC forwards the address, transaction type, TID, and rope number to the RQC. It then signals to the BIB that the transaction "hit" in the IOCC's address space. During the snoop phase the BIB indicates it will defer the transaction by asserting DEFER# and VSBL. The IOCC will write the transaction information (including the TID) to the target rope inside the RQC. The RQC sends the transaction out over the rope to the rope guest. The RQC stores the TID in it's local TID FIFO. The rope guest stores the transaction information in the Delayed Request FIFO. This allows posted write transactions to pass "delayed" transactions for the purpose of deadlock avoidance. The rope guest may attempt the read transaction multiple times due to a target retry or disconnect prior to completion of the transfer. The rope guest is required to maintain fair arbitration between posted write transactions and delayed transactions.

When all the requested data has been obtained, the rope guest collects the data and it is sent over the rope to the zx1 mio. The RQC in the zx1 mio gets the data and combines it with it TID before sending it on the IOCC. The IOCC signals the BIB that it has PIO return data available and eventually the BIB will indicate that the data can be driven directly to the Itanium-2 bus pads and on to the original requester.

Flow control is implemented in the same manner as for MMIO writes and was described earlier in this chapter.

You're probably asking yourself, "Why is the rope guest required to allow posted writes to pass delayed transactions?" The reason is the deadlock avoidance strategy for PCI assumes that completion of MMIO writes is never conditional upon completion of anything else. If a read is ahead of an MMIO write, and the read is retried by the target, the write must still be allowed to proceed. Moving the read to the Delayed Request FIFO allows that to happen.

3.4.1.2.1 Register reads

Register reads are a special case of MMIO reads. As explained in the register write section, there are 2 types of register accesses, zx1 mio register reads and rope guest register reads. For zx1 mio register reads the transaction is sent to the regbus by the IOCC. When the read data is available, the IOCC signals the BIB that read return data is ready. The BIB signals the IOCC when it needs to drive the data directly to the Itanium-2 pads and on to the original requester. Register reads are required to go through a delay transactions FIFO (just like MMIO reads) for deadlock avoidance. Rope guest register reads are handled in the zx1 mio just like an MMIO read.

The register bus in the zx1 mio is actually implemented as a 16-bit daisy-chained connection between all the blocks in the zx1 mio. The address and data must propagate around the entire chain before data is returned. Therefore, register reads tend to be very slow in the zx1 mio.

3.4.1.3 I/O Port Writes

I/O port Write transactions are never posted by the zx1 mio. They operate like a hybrid of MMIO writes and MMIO reads. Data is transferred from the Itanium-2 bus to the PCI bus, just like an MMIO write. However, a completion response is sent from the PCI bus back up the rope to the IOCC, just like an MMIO read.

The IOCC decodes the Itanium-2 bus transaction as an I/O port write and determines the target rope. It then signals to the BIB that it is claiming the transaction and forwards the address, TID, and target rope number to target rope inside the RQC. This is like an MMIO write except the TID is sent along with the other transaction information. As with an MMIO read, the BIB responds during the snoop phase with a deferred response by asserting DEFER#. When the data phase occurs on the Itanium-2 bus the data is claimed by the BIB and the IOCC is told that it owns the data. An I/O port write transaction then proceeds exactly like an MMIO write until the transaction reaches the rope guest. At this point the rope guest stores all the transaction information and data into the Delayed Request FIFO (just like an MMIO read). When the IO-port write reaches the head of the Delayed Request FIFO, the rope guest does a IO-port write transaction. After the rope guest successfully completes the IO-port write on the I/O bus, the transaction behaves as an MMIO read return with no data. The data makes its way to one of zx1 mio's RQC's which combine the data with the TID and forward it to the IOCC. The IOCC signals to the BIB that PIO return data is available and the BIB eventually completes the write using a deferred reply transaction on the Itanium-2 bus.

3.4.1.4 I/O Port Space Read

I/O Port Reads work just like MMIO Memory reads except that on the I/O bus the rope guest will use I/O port transactions instead of the memory read transactions.

3.4.1.5 Accessing I/O Port Space through MMIO

The zx1 mio provides a way of generating I/O port transactions which allows a full 64KB of I/O port space addresses to be allocated to each rope and allows the OS to assign user permissions to I/O port space. (Called GMMIO IO-port space.) From the Itanium-2 bus perspective, GMMIO IO-port transactions are simply MMIO transactions directed to the first 64 MB of MMIO space above 4 GB that is allocated to a particular rope or I/O bus. When the IOCC sees an PIO transaction to an address in this range, it will respond as it does to an I/O port space transaction. The rope guest treats the transaction like an I/O port transaction except that it reduces the address from a 64 MB space to a 64 KB space. This is done by shifting bits 25:12 to 15:2. The PIO transaction supplies 8 byte enables, but an I/O port transaction can only have 4 byte enables. If any of the 4 lower byte enables are asserted, they will be used for the transaction. If not, the upper 4 byte enables will be used. This means that if some byte enables were asserted in both groups, the upper 4 byte enables would be ignored. The response is returned up the rope to the

zx1 mio (ala I/O port writes). Upon seeing the response, the zx1 mio will complete the transaction with a deferred reply transaction.

Regardless of how I/O port space is accessed, all I/O port transactions will be deferred on the Itanium-2 bus until the transaction has completed on the PCI bus. This means that *all* I/O port transactions are non-posted.

3.4.2 DMA

DMA refers to transactions initiated by I/O devices to a memory space which addresses actual main memory. If they address MMIO space it is considered a peer-to-peer transaction (and not DMA). Peer to peer is discussed in a separate section.

3.4.2.1 DMA Memory Read transactions

When the rope guest sees an I/O bus read command to an address that is not in MMIO space, it claims the transaction. The rope guest generates a DMA read request and sends it up the rope to the zx1 mio. The hint information for the request is obtained from a hint table in the rope guest that is indexed by the master ID and one or two address bits. If the address falls within the range that the I/O TLB in the zx1 mio is translating, then the address is translated by the I/O TLB into a 50-bit physical address. If the address is outside the translation range or if the DVI bit is set the address is passed unmodified through the TLB. Once through the TLB, the request will be issued to the Itanium-2 bus if it is a coherent request or directly to the memory controller (via the BIB) if it is a non-coherent request.

When data is returned in response to the read request (either from the Itanium-2 bus or from memory) it is forwarded to the requesting rope and eventually gets to the rope guest. If the DMA device is on traditional PCI the data will be returned immediately to the DMA initiator (connected transaction). Rope guest's are not allowed to retry a PCI read request unless it has a delayed completion resource available. If the request came from PCI-X then the returning data will be queued until there is an opportunity to return it.

The zx1 mio will continue to fetch additional data and send it down the requesting rope at a rate determined by the hint bits, the data width configuration of the rope, and the rate of data consumption by the requesting device. If the request is a fixed-length transaction then it will continue to return data until the specified amount has been returned. If it is a connected transaction then the zx1 mio will continue to return data to the rope guest until it receives a disconnect command.

3.4.2.2 DMA writes

When the rope guest sees an I/O bus write command to an address that is not in its own MMIO space, it claims the transaction. The address and data are sent up the rope to the zx1 mio. If the address is outside the entire MMIO space and the interrupt space (in zx1 ioa), the transaction is assumed to be DMA. DMA write transactions can be arbitrarily long, so any amount of data can

be sent with a single address. The zx1 mio will accumulate the data until the transaction ends or the data spills over into a new cacheline. At that point an exclusive read request would be issued to the Itanium-2 bus or directly to the memory controller if the request were non-coherent. If every byte of the cacheline is being written then the zx1 mio will not request the actual data (only ownership of the cacheline). Once the zx1 mio has obtained ownership for the cacheline (and possibly the data), the DMA data will be moved into the I/O cache where it effectively becomes part of main memory.

If the last byte of the cacheline was written (or if the aggressive flush hint was asserted) the cacheline will immediately be queued for flushing after the data is written into the I/O cache. Eventually the line will be copied directly from the cache to main memory without being visible on the Itanium-2 bus.

It is possible that a CPU could recall ownership of a line prior to the DMA data being copied into the cache. If this happens, the line will be re-fetched once the data is available for immediate copy into the cache. At that time the IOCC will retain ownership until the data has been moved into the cache, thus insuring forward progress.

3.4.2.3 Burst order for DMA transactions (both read and write)

On the Itanium-2 bus, the zx1 mio's cacheline read requests will never use critical word first capability. (i.e., the address will always be cacheline-aligned.) However, when the zx1 mio sources data for an implicit writeback, critical word first must be supported. The Itanium-2 bus interface block will implement that capability.

When the zx1 mio generates a read request to memory to resolve an I/O TLB miss the critical word first feature of the bus will be utilized. The TLB will always use only a single word of the returning data so by requesting it critical word first it makes it very easy (and quick) the get the target word.

3.4.3 Peer to peer (P2P) transactions

The zx1 mio only supports peer-to-peer transactions between I/O devices on the same rope guest's I/O bus or MMIO space write transactions between rope guests. Devices are considered to be on the same the rope guest's I/O bus even if one or more of them are behind a downstream bridge.

3.4.3.1 P2P Read and P2P Writes (same I/O bus)

These types of peer-peer transactions are easy. The rope guest decodes the address on the I/O bus and determines that it is within the MMIO address space allocated to it. Therefore, the rope guest does nothing, allowing the transaction to proceed between the 2 PCI devices.

3.4.3.2 P2P Writes (different I/O buses)

MMIO space peer-to-peer write transactions can be posted. The rope guest decodes the I/O bus memory write transaction. If it is within MMIO space, but not the MMIO space allocated to this the rope guest, the transaction is claimed. Regardless of the size of the memory write transaction on the I/O bus, the rope guest will break them into 16 byte writes or smaller. Each write carries an address and byte enables with it, so any subset of the bytes can be written. The P2P write proceeds through the zx1 mio and is eventually issued on the Itanium-2 bus. Once the transaction is issued on the Itanium-2 bus it comes back into the zx1 mio, which determines the target rope and forwards the transaction to the appropriate rope guest. The transaction then completes just like an MMIO write.

Peer-to-peer I/O ports space writes, regardless of if they are generated by GMMIO writes or not, are not supported (primarily because they have a response and therefore have most of the issues that P2P reads do).

3.4.4 Interrupt related functionality

Normally, PCI devices issue interrupts by asserting one of their interrupt wires (collectively called INTx#). This form of interrupt is supported by the rope guest via an I/O SAPIC. Additionally, message signaled interrupts are also supported. Finally, support for PC legacy 8259 Programmable Interrupt Controllers (PICs) is provided, through support for the int_ack transaction on various I/O buses.

3.4.4.1 Message Signaled Interrupts (MSI)

A message singled interrupt (MSI) transaction occurs when a I/O device does a Memory Write transaction to the 1MB address range starting at 0xFEE0_0000 (zx1 ioa makes MSI space programmable). The rope guest claims the transaction and forwards them to the zx1 mio as interrupt address and interrupt data. The transaction makes its way to the Itanium-2 bus just like a P2P write. The zx1 mio eventually does an interrupt transaction on the Itanium-2 bus with the same address and data, but using the Itanium-2 bus encoding for an interrupt transaction. +Wire Interrupts

The rope guest also contains logic to convert interrupts received over the INTx# pins to interrupt transactions. The logic to do this is called an I/O SAPIC. When an INTx# pin is asserted, the SAPIC looks up the associated entry in a table. The information in the table essentially specifies the address and data values to use for generating an interrupt transaction. The interrupt transaction is eventually forwarded to the zx1 mio and proceeds as described above.

3.4.4.2 Interrupt acknowledge (Int Ack) transaction

To retain PC compatibility, Itanium-2 is capable of receiving interrupts from 8259 PICs and to generate an IntAck transaction in response to fetch interrupt vector information from the 8259. The IntAck cycle is simply a read; it just happens to be encoded differently. The processor initiates an IntAck transaction on the Itanium-2 bus. The zx1 mio claims the transaction and

defers it. An IntAck is then forwarded to the I/O bus below rope 0 (both PCI and PCI-X provide an Interrupt Acknowledge transaction). The 8259 is restricted to this I/O bus. The IntAck transaction proceeds just like an MMIO read. The rope guest issues an IntAck cycle and returns the result back, just as with a regular MMIO read. The BIB does a deferred reply to return the data to the processor.