PCI Interrupt Controller for Industry Standard PCI-ISA Bus Architecture using PCI-to-PCI Bridge Technology

Ross L. Armstrong

Digital Equipment Corporation (Scotland) Ltd.,
Mosshill Industrial Estate,
Ayr, Scotland. KA6 6BE.

The demand for more Peripheral Computer Interconnect (PCI) device configurations beyond the limit set in the PCI local bus specification has prompted the development of several PCI-PCI bridge solutions. This paper describes a new PCI Industrial Computer Manufacturers Group (PICMG) PCI-ISA bus architecture implementation using Digital Equipment Corporation PCI-PCI bridge technology. Layered PCI bus architectures, PCI interrupt latency implications and performance optimizations for PCI-PCI bridge designs are discussed. Reference will be made to Digital's family of 64-bit Alpha Single Board Computers (SBCs) and PCI-ISA backplanes which have been specifically designed to address multiple, low cost, high performance PCI requirements associated with high speed communications and graphics in embedded applications.

Overview

Digital's Embedded and Real-Time line of business has developed a series of modular computing products supporting an open systems environment based upon the PICMG PCI-ISA SBC standard. Two goals of the Digital Modular Computing Component (DMCC) program were to:

- develop a number of PCI based backplane products that would enable customers to procure and configure industry standard PCI and ISA I/O option cards.

- provide extensive PCI I/O option card slots and maintain optimum bandwidth/ performance for bridged slots

Although the former requirement was a simple undertaking, the latter provided a greater challenge to the platform designers. Reduction in bandwidth, however moderate, could be encountered due to software or hardware inefficiencies i.e. degraded interrupt servicing (latency) or propagation/timing delay due to the bridge implementation.

The choice of Digital PCI-PCI bridge chip adequately meets the required timing specification, however interrupt lines derived from secondary bus devices are not routed through the Digital PCI-PCI bridge chip.

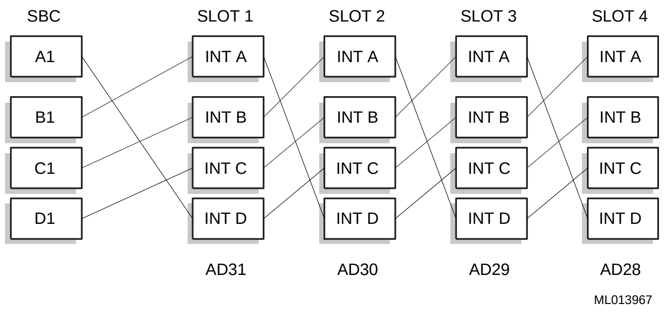
This allowed the designers to thoroughly review and improve upon the standard interrupt binding strategy for PCI buses, where multiple devices might have to share interrupt lines.

The PICMG single board computer connector has only four interrupt lines assigned to it: INTA#, INTB#, INTC# and INTD#, as does each PCI slot connector.

A routing or binding strategy is required to connect between the PCI option I/O and the SBC INTX# line it uses when requesting an interrupt.

In hardware terms, Figure 1, PICMG Single PCI Bus Interrupt Binding, shows how this structure is intended to be provided.

Figure I PICMG Single PCI Bus Interrupt Binding



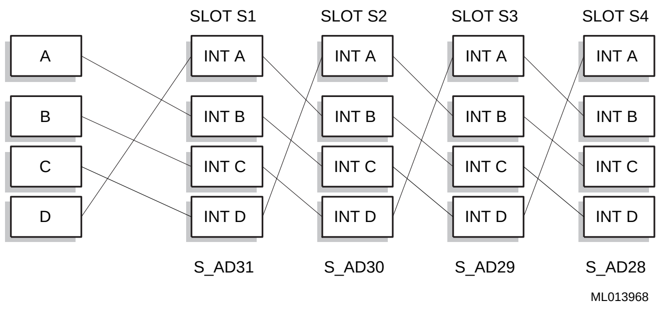
The IDSEI line per slot is assigned to AD[31:28] as per the PICMG Specification. These are used to identify device numbers as given in the configuration address.

The system firmware (or BIOS) code must assume an interrupt binding architecture for its environment. The PICMG specified binding for the primary PCI bus (PCI Bus 0) is shown in Figure 1, i.e. it is hard coded. Because only the firmware (or BIOS) knows how the PCI INTX# lines are routed to the system controller, a mechanism is required to inform the operating system device driver of an interrupt occurrence. This mechanism typically requires a chained 'software' search of each device using a specific hardware interrupt to identify the source.

This can be achieved by having the firmware/BIOS poll all PCI devices to determine which originated the interrupt request and then initiate the correct interrupt service routine, or alternatively, have the operating system kernel interrupt dispatch routine sequentially call each individual device interrupt service routine until the correct source has been identified and serviced. (The latter example is implemented by Microsoft Windows NT).

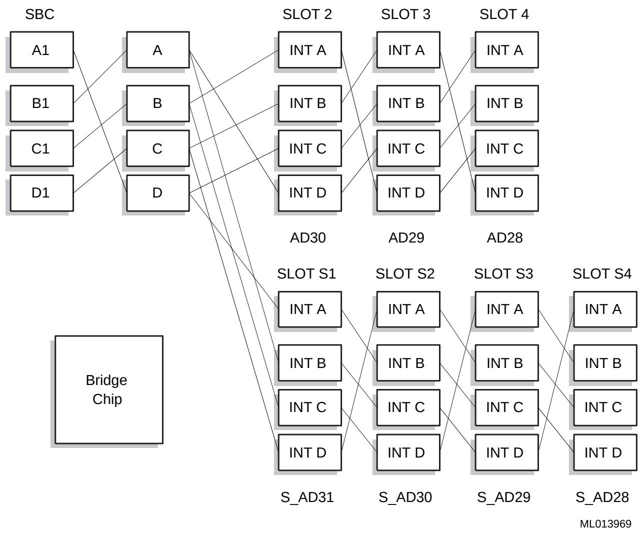
The binding structure becomes even more congested when additional (bridged) PCI buses are implemented in accordance with the *PCI-to-PCI Bridge Architecture Specification Revision I.O.* Their PCI bus interrupts must be connected as per Figure 2, Secondary PCI Bus Interrupt Binding.

Figure 2 Secondary PCI Bus Interrupt Binding



This secondary binding architecture must be overlaid upon the respective primary slot binding that the bridge now occupies. The net effect being illustrated in the example for one PCI-to-PCI bridge in Figure 3, PCI-to-PCI Bridge implementation.

Figure 3 PCI-to-PCI Bridge Implementation



The use of the wire-OR (sometimes also known as *shared*) binding design, means that the polling and decode of an interrupt request can be significantly less than optimal. The latency for this operation is also unpredictable, i.e. with N PCI slots, determining the originator of the PCI interrupt request could take a minimum of 1, up to a maximum of (N-1) bus read cycles.

An alternative solution was investigated in order to improve upon this industry standard binding architecture to avoid compromising PCI interrupt latency performance in large PCI systems. (Any proposal would take cognisance of, and retain support for, the traditional wire-OR scheme.)

The design goal was to provide improved performance while maintaining an open system architecture capable of supporting both existing and alternative modes.

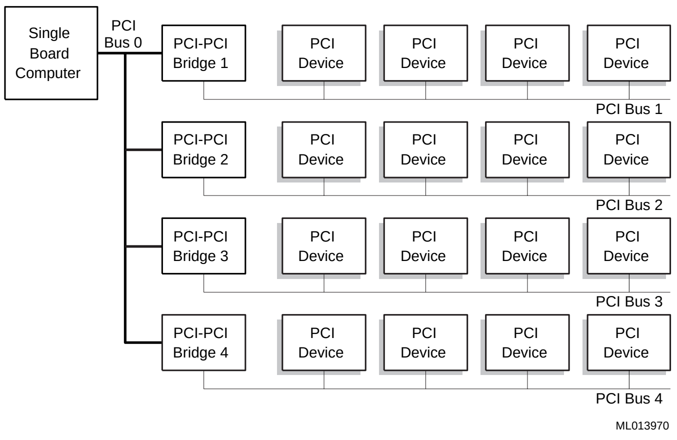
Interrupt Controller

Assumptions/Limitations

The interrupt controller must be able to support up to 4 primary PCI devices, a primary PCI device being either a PCI bridge or a physical connector. Each PCI bridge can have up to 4 secondary devices implemented behind the bridge.

The largest configuration would mean a maximum of 16 individual PCI connectors, as demonstrated in Figure 4, Maximum Allowable PCI Configuration.

Figure 4 Maximum Allowable - PCI Configuration



This implies a maximum of *64 PCI interrupt sources*. A controller that can service all of these product scenarios, or some subset thereof, must do the following:

- •• support up to 4 primary devices
- •
- •• be able to identify whether a primary device is either

    - •• an on-board PCI-PCI bridge (with up to 4 'bridged' secondary connectors behind it) **OR**
    - •• physical connector

- •• be able to uniquely identify each of the 16 potential interrupts that can be generated from a PCI bridged device (i.e. four interrupts from each of the 4 secondary devices) **AND**
- •• be able to uniquely identify each of the 4 potential interrupts that come from a physical connector

This implies that the controller will have the following:

- •• a register (or similar feature) to detail whether a primary device is a physical connector or an on-board PCI-PCI bridge
  •
- •• a register (or similar feature) for each primary device (i.e. 4 in total) to provide status for each PCI interrupt supported by that primary device. Note that the requirements for a bridged device are very different from those for a non-bridged device and the format of the register will be different for each case.
  •
- •• a register (or similar feature) to identify which primary device caused an interrupt (i.e. to prevent having to read all 4 interrupt registers to determine the interrupt source).

The backplanes developed as part of the DMCC program are intended for use with many operating systems and non-Alpha single board computers.

Therefore, they must also be compliant with the shared interrupt scheme as defined in the PICMG *PCI-ISA Card Edge Connector Proposal for Single Board Computer (SBC) Specification, Revision 2.0* and *PCIPCI Bridge Board Edge Connector for Single Board Computer Specification.*

To meet this two fold requirement the proposed controller supports two unique modes of operation with some means of switching between them. For convenience this was determined to be software selectable.

The default mode, at power-up, makes the backplane compliant to the PICMG specification. This will be known as *PICMG Mode.*

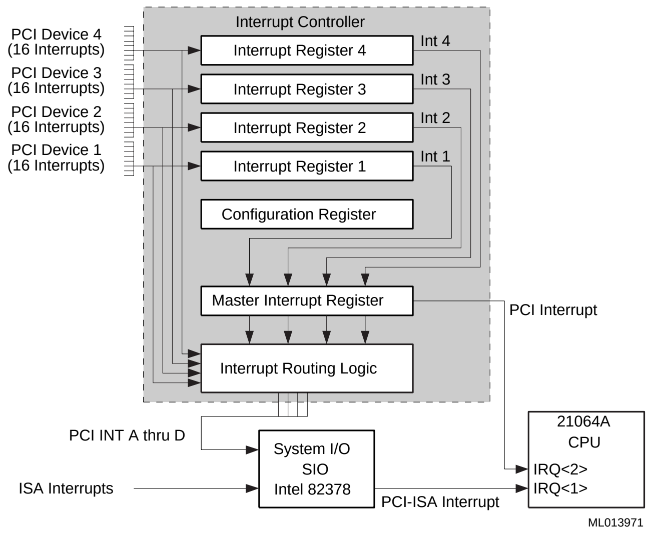
Operating systems (OS) needing to make use of the interrupt controller must explicitly switch, via software, to the desired mode of operation.

A bonus of this design is that the hardware is (in simple terms) a form of hardware interrupt accelerator usable by multiple operating systems and hardware platforms, if their corresponding BIOS or firmware code is appropriately configured. This mode is known as the *Accelerator Mode.*

Generic Architecture

The basic form of the Interrupt controller is shown in Figure 5, DMCC Interrupt Controller Block Diagram. The interrupt controller is split into multiple functional blocks, each section's usage being dependent on the desired mode of operation; either PICMG Mode or Accelerator Mode. These modes are mutually exclusive and are discussed in the following sections.

Figure 5 DMCC Interrupt Controller Block Diagram



The example and illustrations used throughout this paper refer to a specific Alpha 21064A PICMG PCI-ISA single board computer implementation. The concepts are generic and can be fully utilised by alternative platforms. In Figure 5, the interrupt controller functionality is shown within the shaded area and is physically located on the backplane; the System I/O and CPU are resident on the actual SBC.

PICMG Mode

This is the default mode for the Interrupt Controller on power-up. The Register Logic blocks are disabled and all inputs are fed into the Interrupt Routing Logic block. It implements the necessary binding to be compliant with the appropriate PICMG specification (as per figures I & 2) and addresses the routing for 64 individual interrupt request lines to 4 outputs, i.e. the four SBC INTX#. When in this mode, the Interrupt Controller appears as shown in Figure 6, DMCC Interrupt Controller Block Diagram - PICMG Mode.

Interrupt Registers

The Master Interrupt Register and Interrupt Registers 1 through 4 are not available and have no meaning when accessed (i.e. writes are not stored and reads give indeterminable results).

Configuration Register

The Configuration Register again has no real meaning in this mode, however it is always active since it is the means to switch to Accelerator mode. See later for details on how this is achieved.

PCI Interrupt Routing

In PICMG mode, PCI device interrupts are taken directly to the Interrupt Routing Logic where they are simply wire-ORed, to provide INTA, INTB, INTC and INTD, as specified in the *PICMG PCI-ISA Card Edge Connector Proposal for Single Board Computer (SBC) Specification, Revision 2.0 and PICMG PCI-PCI Bridge Board Edge Connector Proposal for Single Board Computer (SBC), Revision 1.* These are routed to the System I/O IRQ lines in the demonstration example provided.

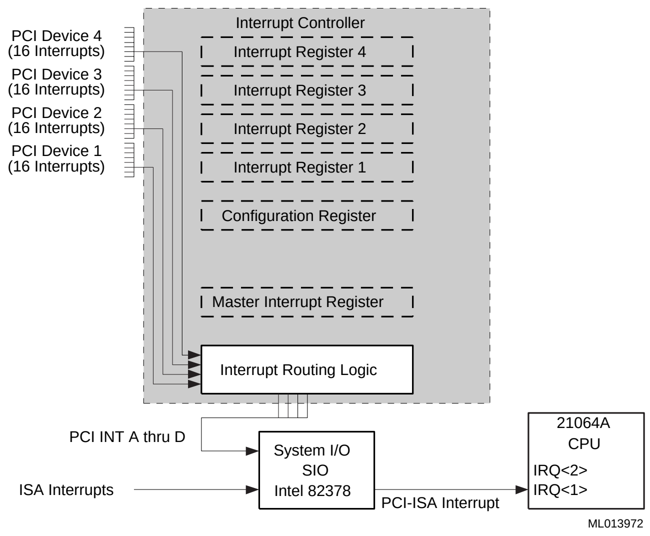


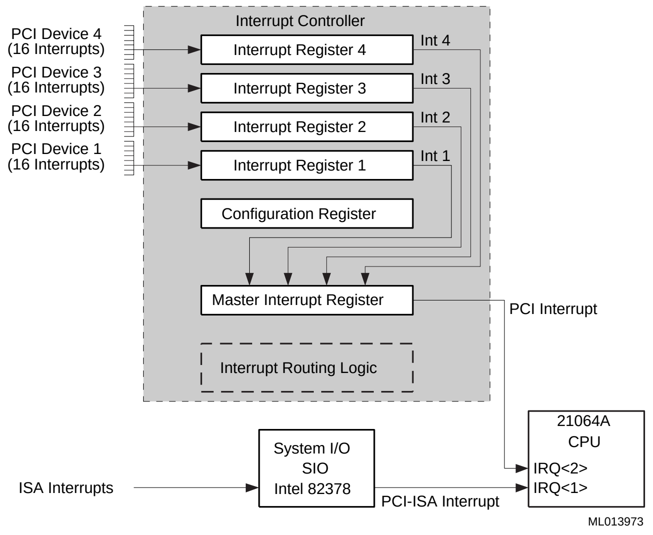
Figure 6 DMCC Interrupt Controller Block Diagram PICMG Mode

Accelerator Mode

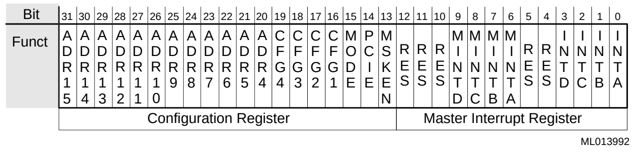
The interrupt controller or Accelerator Mode can be enabled via software only. To enable this mode, the Control Register must be written to, prior to enabling interrupts.

When in this mode the Interrupt Controller looks as shown in Figure 7, DMCC Interrupt Controller Block Diagram - Accelerator Mode.

Figure 7 DMCC Interrupt Controller Block Diagram Accelerator Mode



The software sequence for enabling Accelerator Mode is shown in Figure 8, Accelerator Mode - Software Enabling



Bit	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
Funct	ADR15	ADR14	ADR13	ADR12	ADR11	ADR10	ADR9	ADR8	ADR7	ADR6	ADR5	ADR4	CFG4	CFG3	CFG2	CFG1	MODE	PCIE	MSKEN	RES	RES	RES	MINTD	MINTC	MINTB	MINTA	RES	RES	INTD	INTC	INTB	INTA
	Configuration Register																			Master Interrupt Register												

ML013992

Sequence.

Figure 8 Accelerator Mode - Software Enabling Sequence

All registers are implemented as 32 bit registers addressable in ISA space. (The interrupt controller could as easily have been implemented as a PCI device, however it would then be counted as a full PCI device load and could have had an adverse impact on the total 10-load limit.)

Configuration and Master Interrupt Register

Table 1, Configuration and Master Interrupt Register, defines the register bit allocation. The Configuration Register is always active and is the only means of controlling the Interrupt Controller's behaviour. The Configuration and Master Interrupt Register is located at ISA I/O address 0500h - 0503h.

Apart from having the mode enable bit (MODE), it also stores the high order ISA I/O address bits for Interrupt Registers 1 through 4 (ADR[15:4]). The low order address bits (ADR[3:0]) are fixed at 0000, 0100, 1000, 1100 respectively.

The backplane configuration details are stored in CFG[4:1], (bits [19:16] of the Configuration Register), defining which primary PCI slots are connectors and which are bridge chips i.e. which have four versus sixteen potentially active interrupt lines.

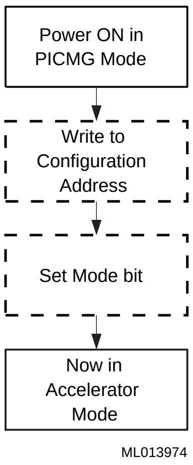


Table I Configuration and Master Interrupt *Register*

In *PICMG Mode,* the PCIE bit defines whether the four INTX# interrupts are routed to the System I/O or whether the one PCI interrupt line is routed directly to the CPU. In typical PICMG applications ISA interrupts are heavily used and the PCEE bit can free up to four ISA interrupts. It is always set in *Accelerator Mode.*

MSKEN is used to support interrupt polling. When enabled, the interrupt status bits in the four interrupt registers are dependent upon their corresponding MASK bits. When disabled, they match the status of the interrupt source.

The Master Interrupt Register is only enabled when in accelerator mode. This register gets its input from the 4 Interrupt Registers and is used to determine which Interrupt Register should be read to find the source of the PCI interrupt.

INT[D:A] reflects the status of the corresponding Interrupt Register[4:1], i.e. INTD status is the logical ORing of the sixteen Interrupt Status bits stored in Interrupt Register 4. INT[D:A] can be correspondingly masked by MINT[D:A].

In this way the PCI interrupt source can be determined in two ISA read cycles; one to the Master Interrupt Register and one to the specific Interrupt Register.

Interrupt Registers[4:1]

Each of the 4 Interrupt Registers represents a primary PCI device. The following table maps the primary PCI device to it's associated Interrupt registers. The address of these interrupt registers is defined by the contents of the ADR bits in the Configuration Register.

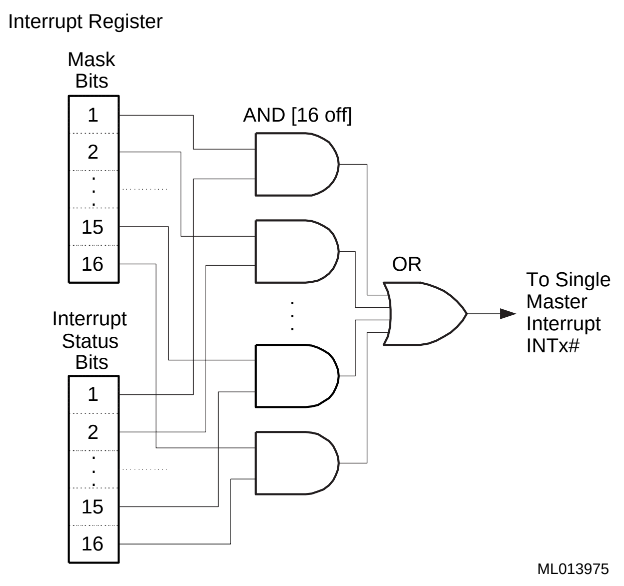
Table 2, Interrupt Register Mapping, defines the Configuration Space Address for each of the primary PCI devices and gives an example of possible ISA I/O addresses for each Interrupt register.

Table 2 Interrupt Register Mapping

The exact format of each interrupt register is dependent on whether a primary PCI device is a physical PCI connector or a PCI-PCI bridge. The format of Interrupt Register 1 through 4 is defined by the CFG bits within the Configuration Register. This can be used to determine the exact configuration of the backplane, and hence the number of potentially active interrupt lines. Table 3, Interrupt Registers[4:1], defines the register bit allocation.

Table 3 Interrupt Registers[4: 11].
If the primary PCI device is a connector, the Interrupt Register stores only the status and MASK bits for four interrupt lines, i.e. only bits [3:0] and [19:16] have any meaning.

Figure 9 Interrupt/ Mask Operation



When the primary PCI device is a PCI-PCI bridge, the corresponding Interrupt Register must store the status of up to 16 interrupt lines for 4 secondary connectors implemented behind the PCI-PCI bridge and also the MASK bits for each individual interrupt line (i.e. all [31:0] bits are valid).

The interrupt STATUS bits [15:0] are ANDed with their corresponding 16 Interrupt Register MASK bits. The results of each AND operation are then ORed together to form a single INTX# signal that is routed to the Master Interrupt Register, as illustrated in Figure 9 Interrupt / Mask Operation.

Primary PCI Device	Associated Interrupt Register	Primary PCI Device Config. Space Address	Interrupt Register ISA I/O Address
1	1	AD31	0510h-0513h
2	2	AD30	0514h-0517h
3	3	AD29	0518h-051Bh
4	4	AD28	051Ch-051Fh

ML013993

Note : If a multifunction option card (i.e. an option with a bridge) is plugged into a physical connector, there is support for the 4 primary interrupts from behind its on-

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Funct	MS4ID	MS4IC	MS4IB	MS4IA	MS3ID	MS3IC	MS3IB	MS3IA	MS2ID	MS2IC	MS2IB	MS2IA	MS1ID	MS1IC	MS1IB	MS1IA

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Funct	S4ID	S4IC	S4IB	S4IA	S3ID	S3IC	S3IB	S3IA	S2ID	S2IC	S2IB	S2IA	S1ID	S1IC	S1IB	S1IA

ML013994

board bridge. If more than four interrupts are used (i.e. via sharing), they are not supported.

Accelerator Interrupt Decode

Hardware Interrupt Architecture

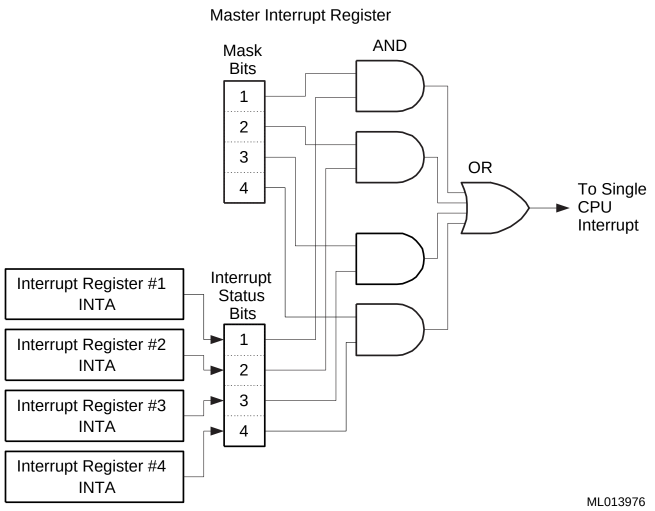
In Accelerator Mode, INT[D:A] in the Master Interrupt Register reflects the status of the corresponding Interrupt Register[4:1] (i.e. INT[D:A] status is the logical ORing of the sixteen Interrupt Status bits stored in each Interrupt Register [4: 1]).

This two stage interrupt register strategy allows rapid decoding of the interrupt source without expanding any individual register set beyond 32 bits.

Figure 10 Interrupt Decode Schematics

The INT[D:A] interrupt status bits are ANDed with their corresponding 4 Master Interrupt Register MASK bits. The results of each AND operation are then ORed together to form the PCI interrupt request signal that is routed to a single IRQ on the CPU, as illustrated in Figure 10 Interrupt Decode Schematics.

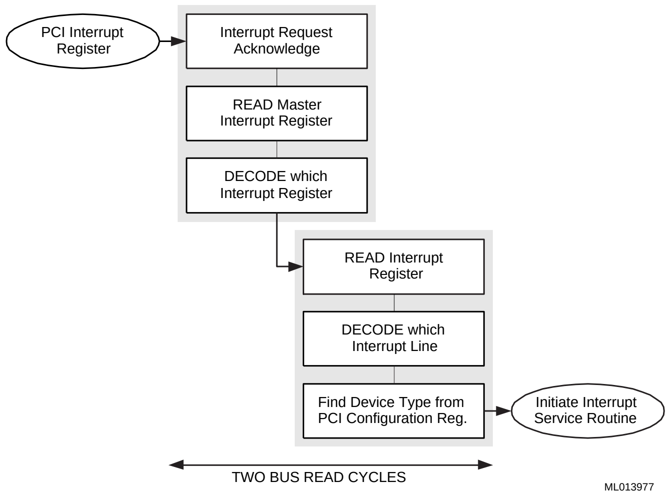
The final logical routing of the Master Interrupt register is not limited by the MSKEN bit status. Routing of the INT[D:A] interrupts is determined only by the value of the interrupt status bit and it's corresponding mask bit.


Master Interrupt Register
Mask Bits
1
2
3
4
AND
OR
To Single CPU Interrupt
Interrupt Register #1 INTA
Interrupt Register #2 INTA
Interrupt Register #3 INTA
Interrupt Register #4 INTA
Interrupt Status Bits
1
2
3
4
ML013976

Firmware/BIOS Interrupt Decode

The interrupt accelerator decode architecture influences the host CPU firmware/BIOS, and is usually transparent to the target operating system. The firmware/BIOS must implement a decode routine as per Figure I 1 Software Decode Steps.

Figure 11 Software Decode Steps - Accelerator Mode


PCI Interrupt Register
Interrupt Request Acknowledge
READ Master Interrupt Register
DECODE which Interrupt Register
READ Interrupt Register
DECODE which Interrupt Line
Find Device Type from PCI Configuration Reg.
Initiate Interrupt Service Routine
TWO BUS READ CYCLES
ML013977

The predictable and repeatable time to dispatch the appropriate interrupt vector (service routine), after receipt of an interrupt request, is two bus read cycles. The decode operation logically occurs in parallel with the read cycle.

Interrupt Latency

The interrupt *dispatch latency* is the elapsed time from receipt of an interrupt request t-o dispatch to the interrupt service routine.

The interrupt *service latency* is the elapsed time from entry of the interrupt service routine to its completion.

Exact interrupt latency, for a given system configuration, will be operating system dependent (i.e. the interrupt *service latency* may vary significantly between operating systems even when the *dispatch latency* in firmware/BIOS is identical).

Operating systems vary in interrupt service routine efficiency and can be equally dissimilar across hardware platforms. The interrupt accelerator optimises the hardware aspect of this process.

PICMG Mode

The wire-ORed binding strategy is not optimal in large PCI slot configurations and most PCI-PCI bridge implementations.

It directly impacts the achievable *interrupt dispatch latency*, and some interrupt service methodologies (e.g. round robin, etc.) can further reduce efficiency in these types of environments.

The dispatch latency is also *unpredictable* i.e. with N PCI slots, determining the originator of the PCI interrupt request could take a minimum of 1, up to a maximum of (N-1) bus read cycles.

Figure 12 Software Decode Steps - PICMG Mode

The interrupt *dispatch latency* in very large systems can result in severe degradation of the system performance. This is caused by I/O devices 'stalling' because they cannot get serviced efficiently. In extreme configurations, a particular device may 'never' get its interrupt serviced, resulting in failure of that functionality (e.g. a network card may 'drop' in-coming packets or a serial line may 'drop' received characters).

Accelerator Mode

The accelerator architecture offers *predictable* and consistent interrupt *dispatch latency* resulting in higher performance for large PCI configurations. *Predictability* is key in most real-time applications.

The corresponding accelerator interrupt *dispatch latency* is always two Bus read cycles.

Physical Implementation

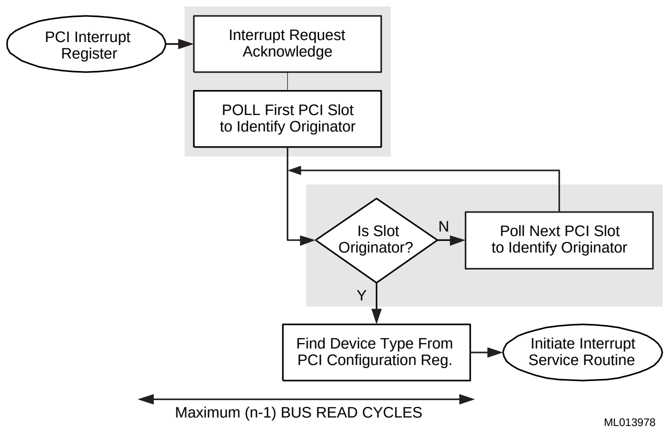
This paper is not intended to imply any particular physical implementation. The generic functionality for a PICMG application can be implemented either as an ISA or PCI based device, however it could also be supported in alternative bus architectures.

Summary

Standard motherboard implementations provide PCI interrupt binding (for the firmware/BIOSI decode in the physical etch routing).

This binding structure becomes congested when additional (bridged) PCI buses are implemented in the system. This means that the polling and decode of an interrupt request can be significantly less than optimal. The interrupt latency is also unpredictable.

The proposed interrupt accelerator design described in this paper results in the *predictable***,** repeatable (consistent) and improved interrupt dispatch latency, *key for real-time applications***.**


PCI Interrupt Register
Interrupt Request Acknowledge
POLL First PCI Slot to Identify Originator
Is Slot Originator?
N
Poll Next PCI Slot to Identify Originator
Y
Find Device Type From PCI Configuration Reg.
Initiate Interrupt Service Routine
Maximum (n-1) BUS READ CYCLES
ML013978

Acknowledgements

The DMCC program was the cooperative effort of a large number of engineers whose initiatives have contributed to this paper. Thanks in particular to Alan Milne, Sean McGrane, Robin Alexander, Vikas Sontakke and John Lenthall, all of whom have worked within the E&Rt engineering design team to ensure the successful product implementation of the concepts detailed in this paper.

References

1. PCI Local Bus Specification, Revision 2.0
2. PICMG, PCI-ISA Card Edge Connector Proposal for Single Board Computers, Revision 2.0
3. PICMG, PCI-PCI Bridge Board Edge Connector Proposal for Single Board Computers, Revision 1.01
4. PCI to PCI Bridge Architecture Specification, Revision 1.0

Author

Ross Armstrong, the Project Leader for the DIGITAL Modular Computing Components (DMCC) program, is a

principal hardware design engineer with the DIGITAL Embedded and Real-Time (E&Rt) engineering Organization. Ross holds a B.Sc. (Eng.) Hon's from Aberdeen University and a joint Master of Technology Management (M.T.M.) from Strathclyde and Heriot Watt Universities.