TECHNICAL
BRIEF

i960® RP Processor: A Single-Chip Intelligent I/O Subsystem

intel®

Additional copies of this document or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641
or call 1 (800) 548-4725

intel®

TABLE OF CONTENTS

INTRODUCTION 1
- Examples of Intelligent I/O Innovation 1
- Storage I/O Interfaces 2
- Network I/O Interfaces 2
- Emerging Technologies 3
- Expanding the Market for Intelligent I/O by Reducing System Costs 3
- Expanding Server Capabilities with an Intelligent I/O Subsystem 3
- Intelligent I/O Processor Requirements 4

i960® RP PROCESSOR TECHNICAL OVERVIEW 5

i960® RP PROCESSOR HIGHLIGHTS 6
- Core Architecture Performance 7
- Fast Call-and-Return Mechanism 7
- Set-Associative Cache Design 8
- Enhanced Bus Control Unit 10
- Superior Interrupt Performance 11
- i960® RP Processor Integrated Peripherals 12

i960® RP PROCESSOR DATA FLOW REQUIREMENTS 13

PCI-TO-PCI BRIDGE 14
- Electrically Isolated PCI Buses 14
- Isolated Data Flow 14
- i960® RP Processor Bridge Address Decode 14
- ISA Address Forwarding 16
- VGA Addressing and Snooping Support 16
- VGA-Compatible Addressing 17
- 64-Bit Addressing with Dual Address Cycle 17
- Bridge Queues 19
- PCI Bridge Transactions 19
- PCI Data Streaming 20

ADDRESS TRANSLATION UNITS (ATU) 21
- ATU Queues 23
- ATU Inbound Transactions 23
- Inbound ATU Data Streaming 23
- Outbound ATU Transactions 23
- ATU Direct Addressing Transactions 24
- Generating PCI Configuration Cycles 24

intel®

PRIVATE PCI DEVICES 25
Private PCI Address Space 25
MESSAGING UNIT 27
Message Registers 27
Doorbell Registers 27
Circular Queues 27
Index Registers 29
DMA CONTROLLER 30
Hardware Unaligned DMA Transfers 30
DMA Chaining Operation 30
DMA Channel Queues 31
DMA Transactions 32
Demand Mode DMA Transfers 32
BUS ARBITRATION SUPPORT 33
Local Bus Arbitration Unit 33
Secondary PCI Bus Arbitration Unit 34
Primary and Secondary Internal PCI Bus Arbiters 34
INTEGRATED MEMORY CONTROLLER 36
DRAM Control 36
Programmable Refresh Timer 36
DRAM Performance 38
Memory Controller Error Reporting 39
Programmable Byte Parity for DRAM 39
Bus Monitor Support 39
SRAM, Flash and ROM Control 39
FILTERING PCI INTERRUPTS 42
I/O APIC Interface 42
SERIAL I^2C INTERFACE 44
i960® RP PROCESSOR CLOCKING 45
PCI Configuration 45
Peripheral Programming Interface 45
Reset Configuration Options 46
i960® RP PROCESSOR PACKAGING 47
COMPLETE TOOL SET FOR EMBEDDED DESIGN 48
State-of-the-Art Compiler Technology 48
Integrated Software Debuggers 48

Wide Variety of Operating Systems 48
Emulators and Logic Analyzers 48
Evaluation Platforms 48
i960® Microprocessor PCI I/O Software Development Kit 49
PCI Compliance 49
SUMMARY 49

LIST OF FIGURES

Figure 1. Add-In Card Application 2
Figure 2. Server Slot Extension Application 4
Figure 3. i960® RP Processor Block Diagram 5
Figure 4. i960® JF Processor Core Block Diagram 6
Figure 5. i960® RP Processor Register Model 7
Figure 6. Register Cache Example 8
Figure 7. Instruction Cache Hit Rate Comparison 8
Figure 8. Performance Simulation for Various Cache Sizes 9
Figure 9. Application Code Density 9
Figure 10. Physical and Logical Memory Control Example 10
Figure 11. i960® RP Processor Data Flow Requirements 13
Figure 12. i960® RP Processor Bridging Operation 15
Figure 13. ISA Mode Address Forwarding 16
Figure 14. VGA Compatible Address Forwarding 18
Figure 15. Primary ATU Inbound PCI Address Translation Example 22
Figure 16. Private PCI Address Space 26
Figure 17. Overview of Circular Queue Operation 28
Figure 18. DMA Chaining Example 31
Figure 19. Demand Mode DMA Example 32
Figure 20. Local Bus Arbitration Example 34
Figure 21. Non-Interleaved DRAM Example 37
Figure 22. Interleaved DRAM Example 38
Figure 23. 8-bit Memory Example 40
Figure 24. 32-bit SRAM/Flash Memory Example 41
Figure 25. PCI Interrupt Steering 43
Figure 26. Connecting the IDSEL# Signal Example 46

THE i960® RP PROCESSOR FOR EXTENDING PCI BUS CAPABILITIES AND ENHANCING SERVER PERFORMANCE

INTRODUCTION

Today's client/server computing environment presents a challenge for maximizing server performance. Increasingly powerful host CPUs can be relieved of I/O processing in order to perform at optimal levels. Creating intelligent I/O subsystems, based on high-performance embedded processors, allows the host CPU to perform more effectively.

Other factors driving the need for intelligent I/O subsystems include:

- The stand-alone computing model is being replaced by networked computing.
- Networked computers increase the vast quantities of data the server systems support.
- Since servers CPUs also run user applications, they need more powerful storage interfaces for accessing larger disk storage areas, in addition to higher reliability offered by RAID storage.
- Simultaneously, the data sizes and type increasingly contain natural data elements like video or audio, in addition to text and graphics.

Clearly, in today's client/server model, data congestion occurs more frequently at the servers. An intelligent I/O subsystem creates the balance between server performance and the data I/O paths, thus relieving data congestion.

Examples of Intelligent I/O Innovation

Today's high-performance add-in adapter cards demonstrate intelligent I/O innovation. The adapter card is a subsystem consisting of an embedded processor, memory, and peripheral components. The i960® RP processor provides a high level of PCI integration with an independent PCI bus local to the add-in card. Low-cost PCI peripheral components help reduce system costs. The highly integrated i960 RP processor represents a breakthrough in assisting server host CPUs to attain even higher performance levels. It does so by alleviating host CPU interrupts resulting from I/O service request via the PCI bus. Figure 1 shows the block diagram for an intelligent I/O add-in card using an i960 RP processor.

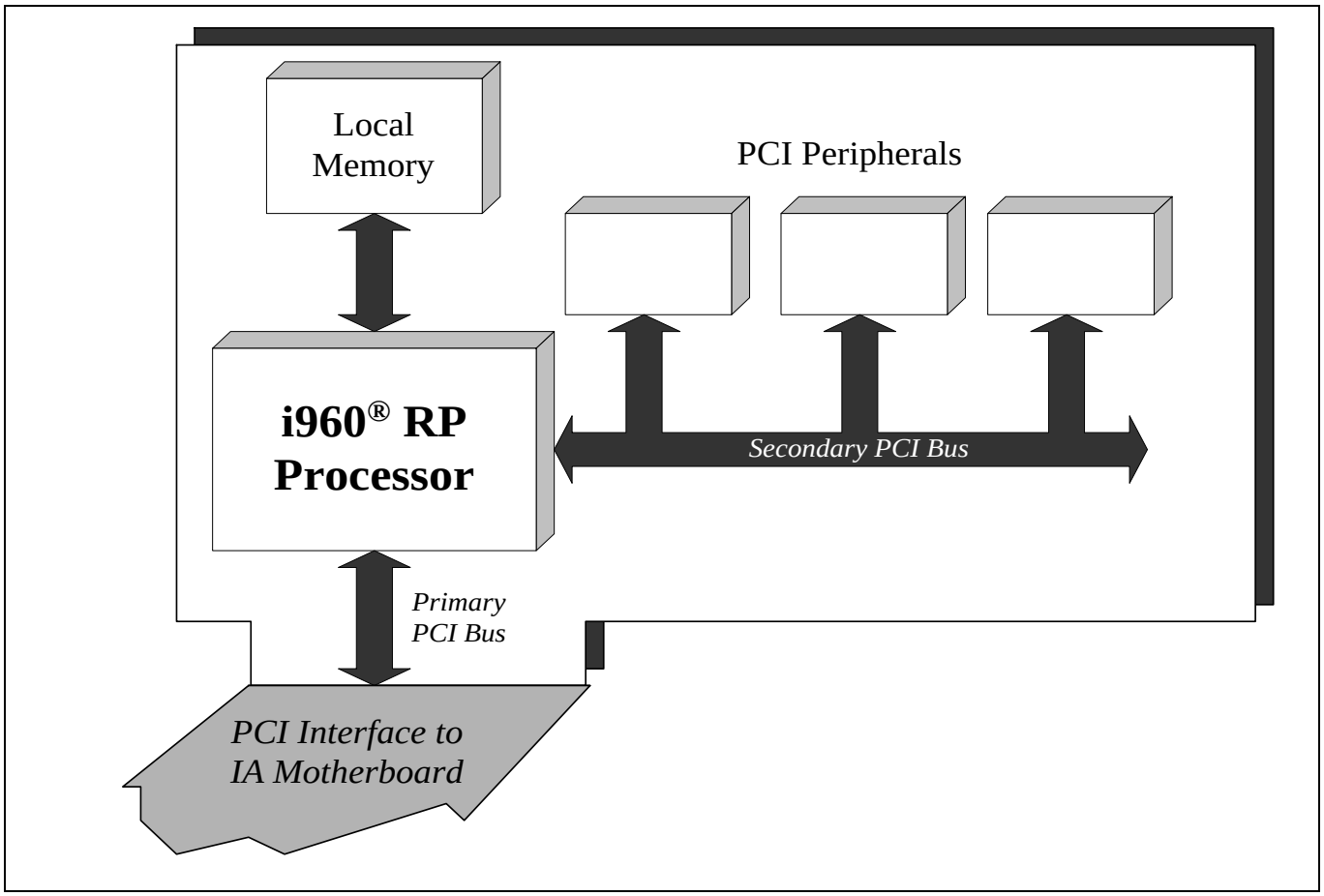

Figure 1. Add-In Card Application

Storage I/O Interfaces

In the two basic areas of server I/O, networking and storage, intelligent I/O subsystems offer enhanced server performance capabilities.

RAID controllers describe one of the better known examples of storage I/O. The server can initiate a disk *store* or *retrieve* command as if it were writing to a single disk. The intelligent RAID controller separates commands into parallel *read* or *write* commands to its attached array of disks. This parallel operation, controlled by the intelligent I/O processor, compensates for the single disk spin-up delay and provides data redundancy. This results in superior data transfer rates as well as greater reliability, a critical necessity as servers become more widely used for corporate computing and database transactions.

A similar example of how intelligent I/O improves server storage connection is in caching disk controllers. In this application, the host can write a data file to the intelligent disk controller cache at speeds matching fast DRAM memory. The host application execution continues while the I/O processor controls the actual disk storage sequences.

Network I/O Interfaces

On the network I/O side, the servers perform many of the bridging and routing functions. The Ethernet or token ring LAN interface with intelligent processors handle the frequent I/O interrupts and intelligently buffers messages to and from the host. This allows the host to streamline application processing and to use other system resources, such as the system bus and memory, more effectively.

In wide-area network (WAN) interfaces, intelligent I/O processing also offers significant performance improvements. For example, intelligent WAN interfaces can compress large message files prior to transmission. This not only frees the host CPU to perform more valuable application tasks, but can allow more users to share the same fixed, wide-area connection. The host CPU is transparent to this operation, thus providing better utilization of the WAN.

Emerging Technologies

In both storage and network interfaces, new intelligent I/O approaches will shrink the time-to-market for newer, high-performance interfaces. As new interfaces emerge, they are often implemented before an industry-wide agreement on all aspects of the interface. However, server buyers often forego the performance benefits of breakthrough technological advances because they fear costly hardware changes when interface implementations do become standardized.

Intelligent I/O solutions, however, alleviate this fear with the software programmability of the appropriate interface elements. For example, with asynchronous transfer mode (ATM), a next-generation network interface, uses an intelligent adapter architecture to implement flow control. Users take advantage of these adapters while the ATM Forum considers flow control standards. As these standards solidify, software updates provide an easy way to maintain compliance with the evolving standards.

These same benefits apply to storage interfaces. For example, early-adopter Fibre Channel suppliers are speeding adoption by offering intelligent I/O solutions that will evolve to support maturing standards.

Expanding the Market for Intelligent I/O by Reducing System Costs

As the price per million of instructions per second (MIPS) of 32-bit embedded processing has fallen, lower prices have become the catalyst for significant system innovation. Innovators can get leading edge 32-bit embedded processors for prices that are far lower than when their previous-generation EISA or MCA bus products were designed. Because the PCI bus allows greater concurrency, each I/O processor's MIPS provide a more meaningful contribution to overall compute system performance.

Low-cost embedded I/O processing utilizes peripheral components to provide the physical interface to I/O devices. PCI local bus standardization has migrated to the peripheral components, providing components with direct connection to the PCI bus. Utilizing these components with an embedded I/O processor creates the intelligent I/O subsystem.

This unique combination of a robust, widely accepted PCI bus and low-cost embedded I/O processing creates a tremendously fertile environment for I/O innovation.

Expanding Server Capabilities with an Intelligent I/O Subsystem

The PCI local bus standard enforces a maximum number of electrical loads that can be connected to the PCI local bus. This drives the requirements for PCI-to-PCI bridges within the server to expand the number of card slots. For example, a PCI bus operating at 33 MHz allows a maximum of ten electrical loads on the PCI bus. Each card slot appears as two electrical loads (one electrical load for the PCI connector and one electrical load for the PCI card) within the server. Today's servers support up to three PCI card slots. Expansion for additional card slots requires a PCI-to-PCI bridge to create a hierarchy of electrically isolated PCI buses.

The i960® RP processor increases the number of server expansion cards slots, in addition to creating an intelligent I/O subsystem. Figure 2 shows an example of the server slot expansion.

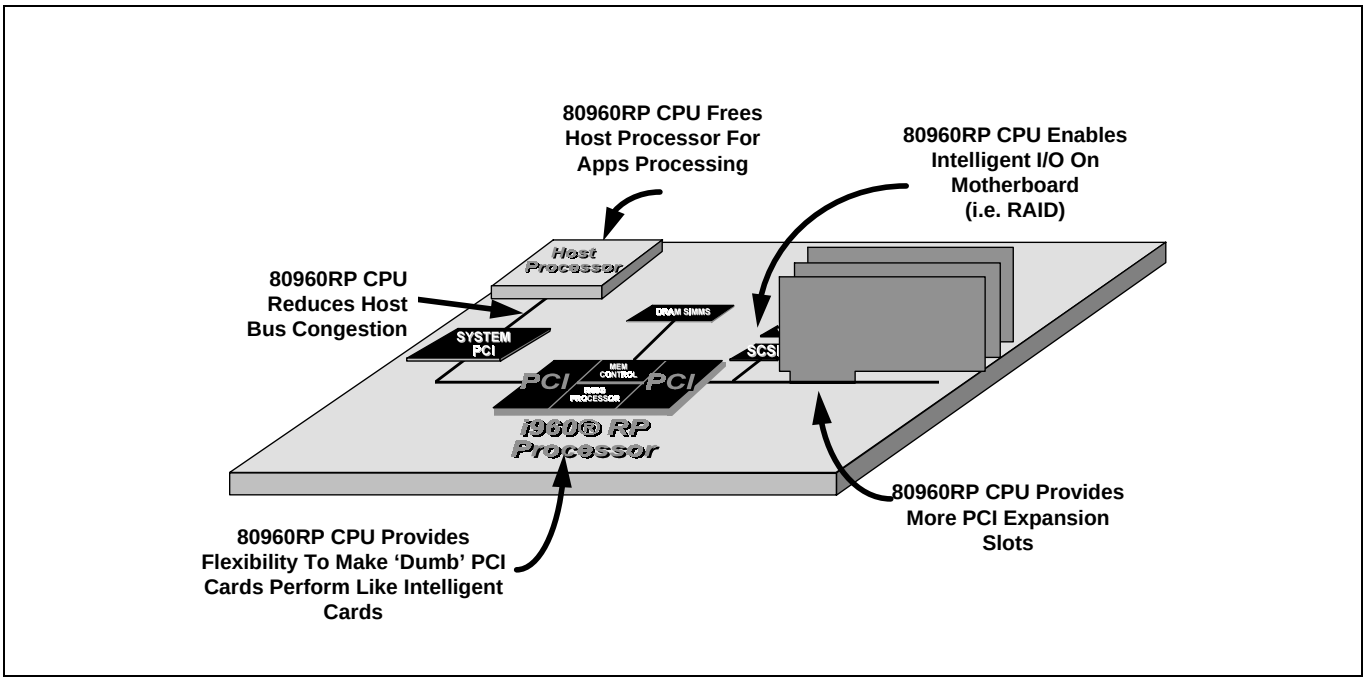

Figure 2. Server Slot Extension Application

Intelligent I/O Processor Requirements

The system requirements for a high-performance Intelligent I/O processor fall into five basic categories. The applications include both server motherboards and add-in adapter cards.

- **Bus Bandwidth** — Maximize bus-throughput & concurrency
  Reduce wait-states
- **PCI Availability** — Isolating data traffic
- **Processing Performance** — Reduce interrupts to host CPU
  Handle I/O processing
- **Data Integrity** — Notification to I/O processor
- **Flexibility** — Create cost-effective, performance driven memory systems

Innovative design solutions match or meet these requirements for an intelligent I/O processor for PCI applications. The i960® RP processor contains hardware features to ease both hardware design and software design.

The i960 RP processor architecture focuses on creating an PCI intelligent I/O subsystem. The 80960RP processor core is the i960 JF processor. Integrated around the i960 JF processor core are peripherals to complete requirements of the intelligent I/O subsystem. The i960 RP processor contains two PCI buses, i960 processor local bus, DRAM Bus, I^2C Bus, and APIC bus. This highly integrated single-chip component enables low-cost intelligent I/O subsystems.

i960® RP PROCESSOR TECHNICAL OVERVIEW

The i960® RP processor integrates functions required for creating an intelligent I/O subsystem. A block diagram of the i960 RP processor, shown in Figure 3, highlights the functionality necessary to provide an intelligent I/O subsystem.

Integrated around the i960 JF processor core are the peripherals necessary to an intelligent I/O subsystem. The basic units of the i960 RP processor include:

- i960 JF microprocessor core
- PCI to PCI bridge
- Address translation unit for direct access between PCI and the 80960 local bus

Messaging unit

- PCI arbiter for secondary PCI bus
- DMA controller
- Integrated memory controller
- I/O Advanced Programmable Interrupt Controller (APIC) interface
- I^2C interface
- Interrupt routing

The following sections describe each of these functional units.

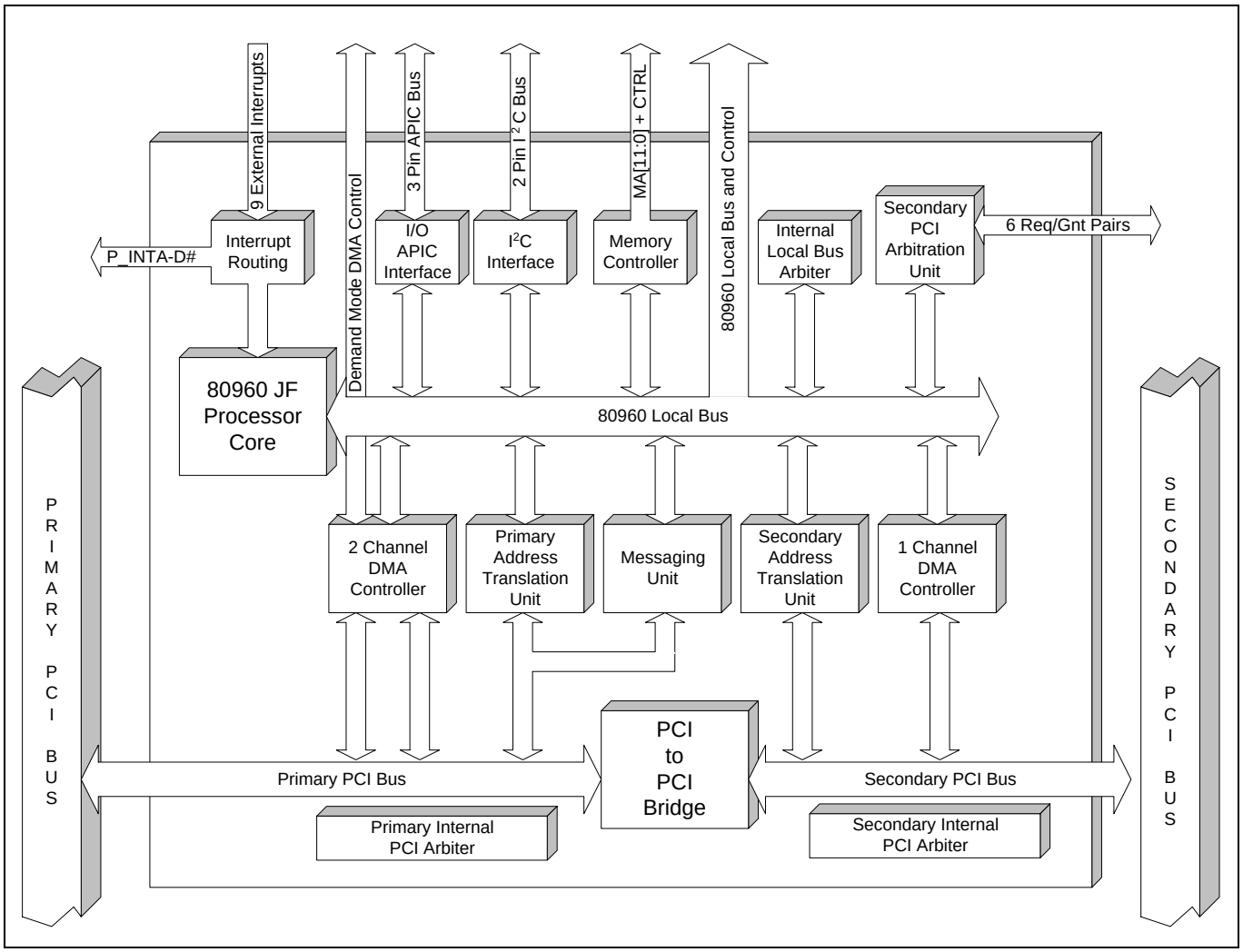

Figure 3. i960® RP Processor Block Diagram

i960® RP PROCESSOR HIGHLIGHTS

The i960® RP processor is a high-performance embedded processor that can achieve more than 31 VAX MIPS[1]. The i960 JF processor core block diagram, shown in Figure 4, highlights many of the features and integrated peripherals.

The i960 RP processor features enhancements to the typical RISC processors that improve performance and time-to-market. These enhancements include:

- Core Architecture Performance
- Fast Call-and-Return Mechanism
- Set-Associative Cache Design
- State-of-the-Art Testability
- Integrated Peripherals
- Flexible Interrupt Controller
- Interrupt Performance Optimizations
  - Caching of Interrupt Vectors
  - Dedicated Register Sets
- - Cache Locking Mechanism
- Enhanced Bus Control Unit
  - Variable Bus-Width Support
  - Flexible Data Caching Mechanism

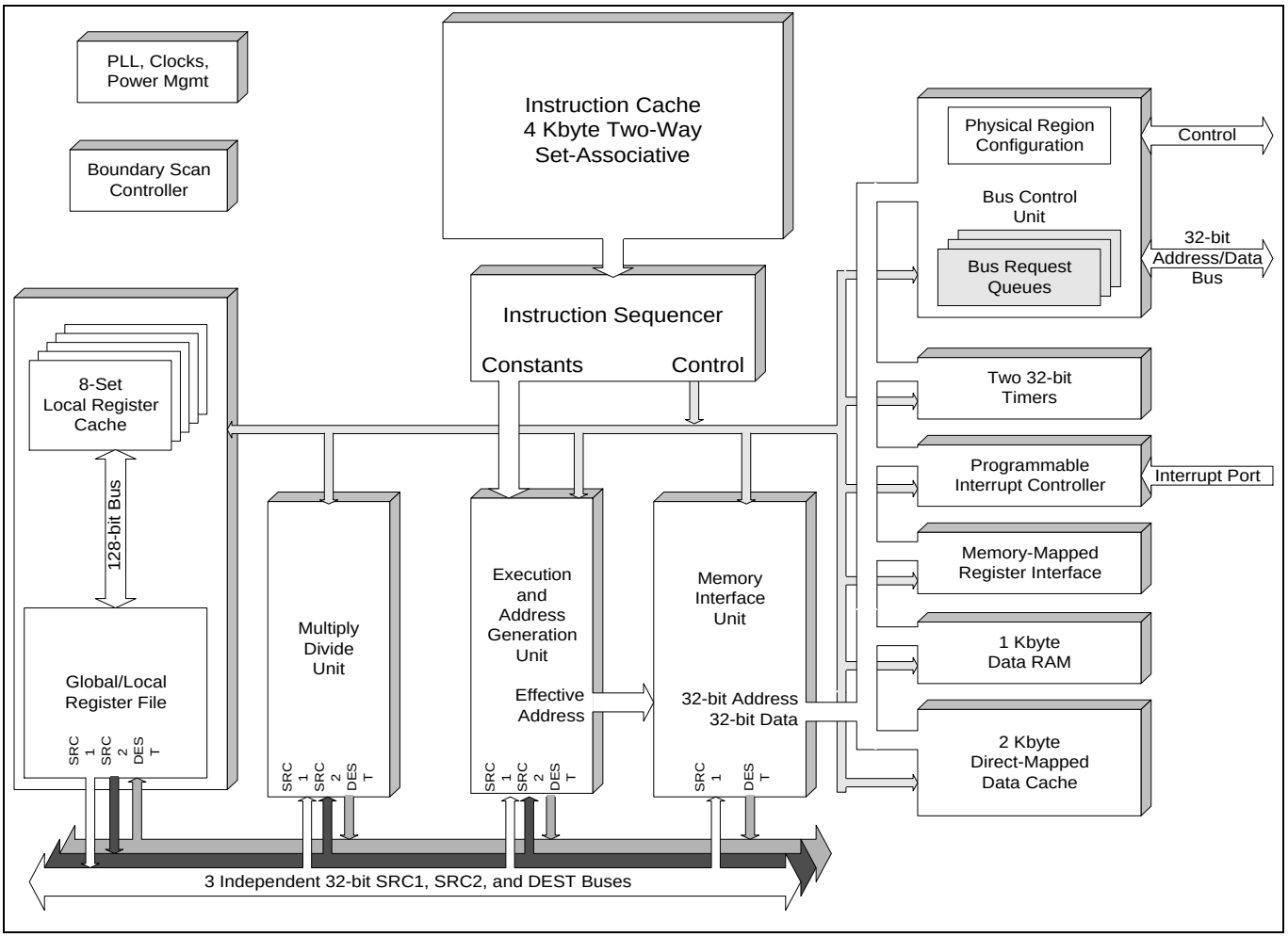

Figure 4. i960® JF Processor Core Block Diagram

[1] Based on Dhrystone Version 2.1 benchmark.

Core Architecture Performance

The i960 RP processor features an independent execution-unit (EU); a multiply-divide-unit (MDU); and a load-store-unit. Individual register scoreboarding, a feature unique to the i960 architecture, enables the processor to execute some instructions in parallel or out-of-order. In addition, the 80960RP core supports all combinations of register source and destination bypassing, a mechanism which allows these execution units to pass the results back to the register file. Register scoreboarding and bypassing enables the processor to sustain single-cycle instruction execution.

Fast Call-and-Return Mechanism

Software performance is crucial in an embedded microprocessor environment. The i960 architecture features a fast call-and-return mechanism to support the modular software base found in today's applications. The local register cache provides storage for the local registers available to software applications. An executing program always has access to sixteen 32-bit local registers and sixteen 32-bit global registers. Executing a procedure **call** or return (**ret**) instruction preserves the contents of the global registers across procedure boundaries. The core allocates a unique set of local registers for each new procedure and deallocates them on the return from that procedure. Figure 5 illustrates the available registers and the register cache.

Execution of a single **call** instruction saves the local registers and allocates a new stack frame for the new procedure. The return (**ret**) instruction deallocates the current local register set and restores the previous state of the machine. The core allocates and deallocates each of these register sets in the local register cache embedded within the processor core. This mechanism provides for fast movement of registers to and from the internal local register cache over a 128-bit wide bus. This feature allows the core to save or restore all sixteen local registers in four clocks.

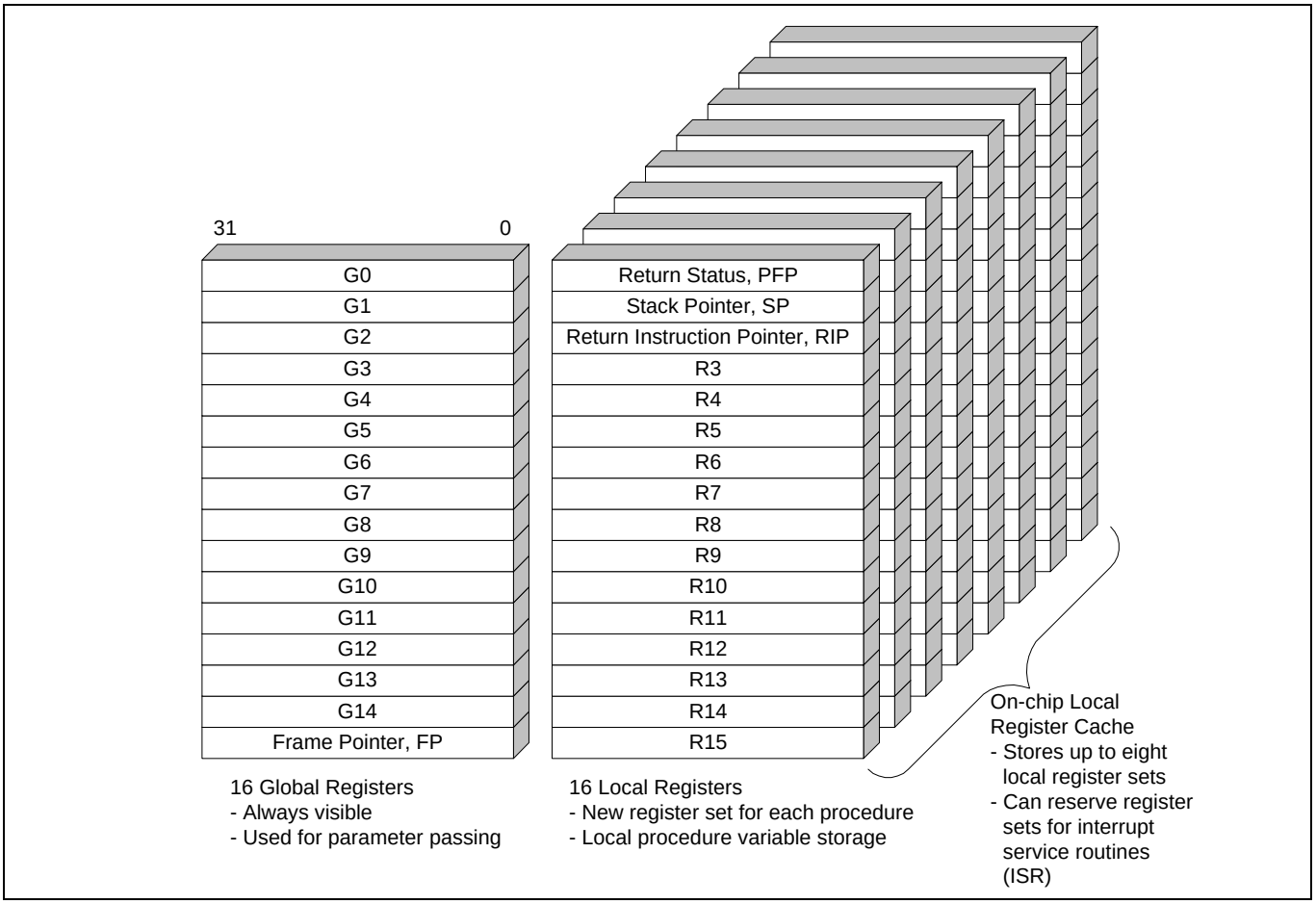

Figure 5. i960® RP Processor Register Model

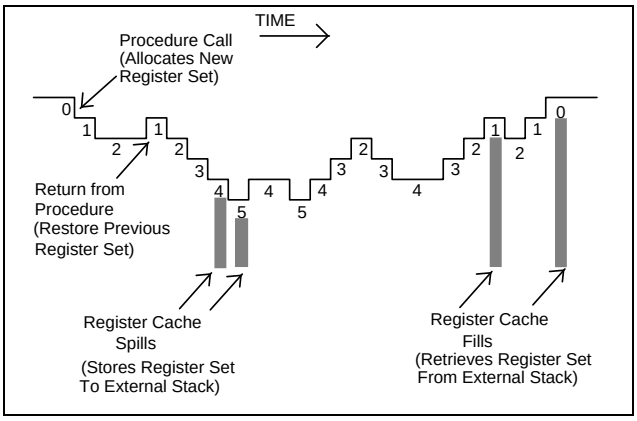

Figure 6. Register Cache Example

The i960 RP processor provides enough storage in the local register cache to maintain eight nested **call**s. When the call mechanism needs additional register sets, the processor will automatically spill (or write to external memory) the oldest register set in the local register cache. This provides a fresh set of local registers for the currently called procedure. Execution of a return instruction will automatically fill (or read from external memory) the current register set with the previously spilled registers. In addition, the register architecture allows the application to reserve register sets for high priority interrupts. Figure 6 shows the register cache behavior over time with four register sets reserved for high priority interrupts.

Software complexity for embedded control applications and faster time-to-market pressure forces the embedded system designers to rely on high-level languages in the development of their application. The nature of structured high-level code requires a mechanism to save the state of an executing process when calling a procedure and torestore the execution state upon return from the procedure. Typically RISC architectures leave these duties for the operating system, requiring multiple instructions to save the processor's state. Recognizing the frequent use of the call and return operation, the i960 architecture completely integrates this mechanism on-chip.

Set-Associative Cache Design

The i960 RP processor contains three independent caches; instruction cache, data cache, and local register cache. In addition to the on-chip caches, there is a zero wait-state on-chip data RAM.

Cache organization refers to the method of associating locations in main memory with locations in the cache. Most embedded RISC processors have direct-mapped instruction caches. In a direct-mapped cache, only one specific location in the cache translates into a specific location in main memory. In a two-way set-associative cache, one of the two different locations in the internal cache translates to the specific location in main memory.

Academic studies show that a two-way set-associative cache is typically as efficient as a direct-mapped cache of twice the size[2]. Figure 7 shows the cache hit rates for both types of cache organization.

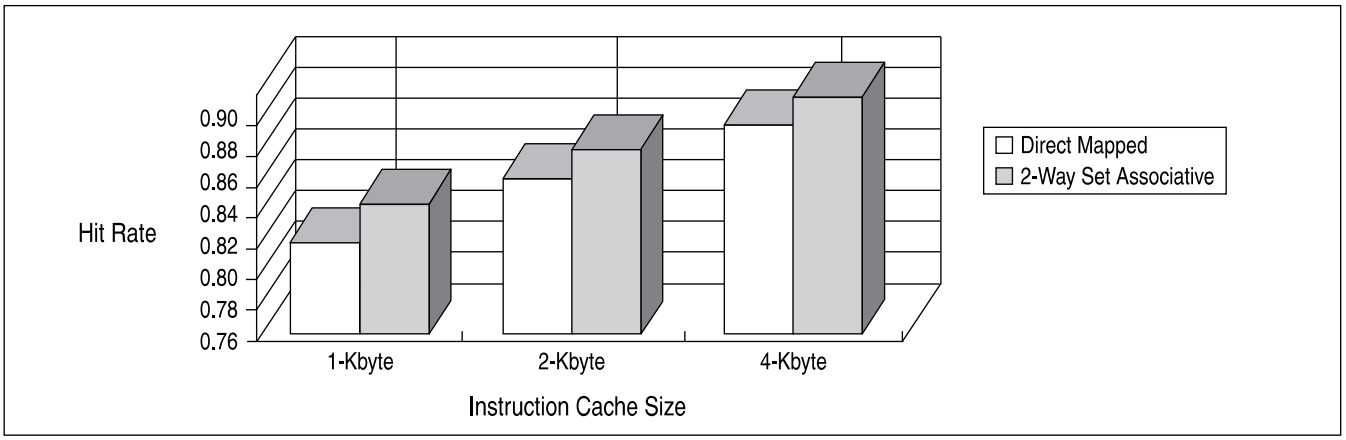

Figure 7. Instruction Cache Hit Rate Comparison

[2] *Computer Architecture A Quantitative Approach*, Hennesy & Patterson, 1990

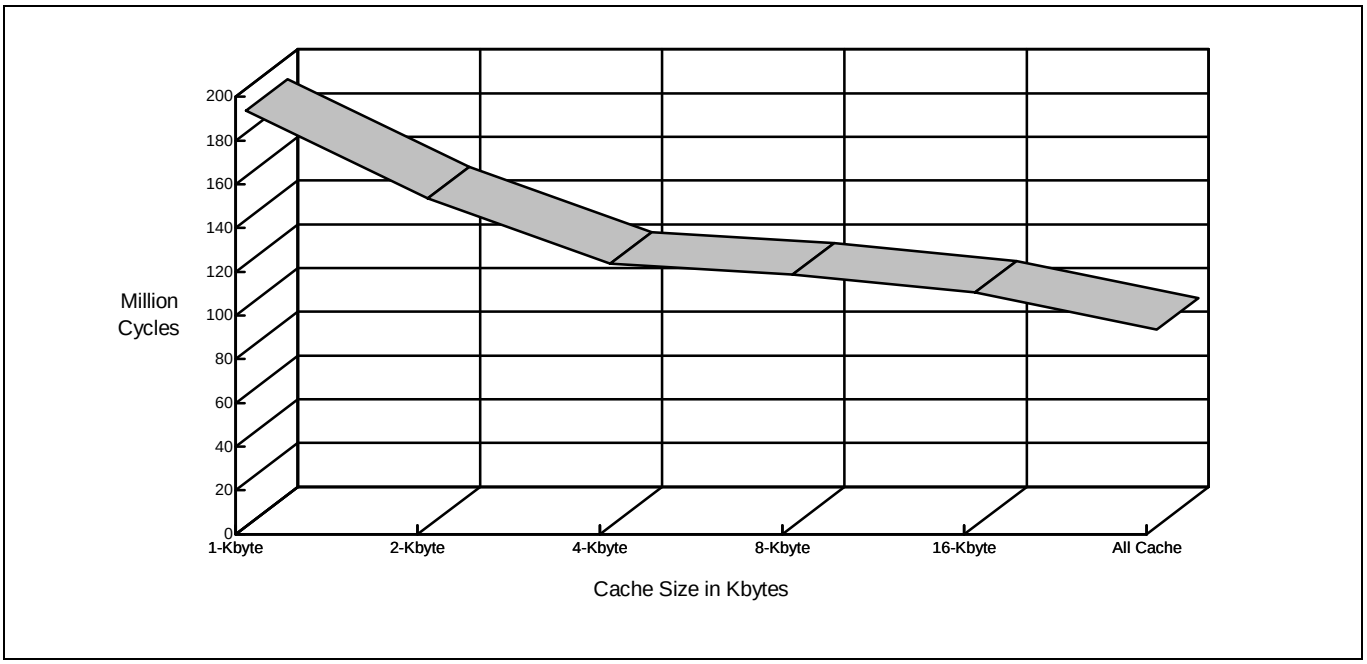

Figure 8. Performance Simulation for Various Cache Sizes

When comparing the hit-rate of a two-way set-associative cache and a direct-mapped cache of twice the size, the processor performance is approximately equal.

By using a 4-Kbyte two-way set associative cache architecture, the designers of the i960® RP processor can provide the efficiency of an 8-Kbyte direct mapped instruction cache while consuming only half the silicon. Through an extensive performance analysis study performed by the Intel 80960 design team, an instruction cache of 4-Kbytes provides optimal performance at an optimized cost.

Figure 8 shows the results for a typical performance simulation.

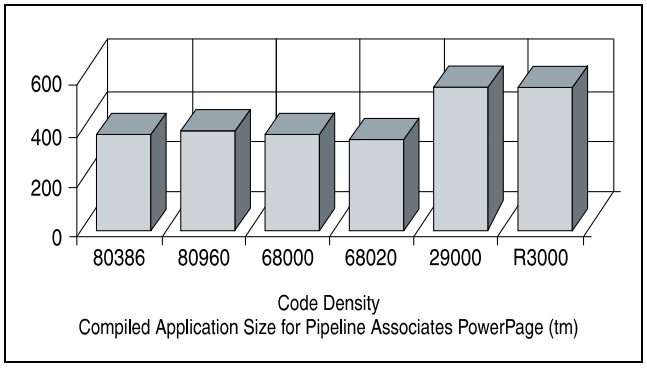

Figure 9. Application Code Density

An important consideration when comparing cache performance is code density. Code density is a relative measure of the size of compiled programs. Programs compiled for the i960 architecture produce code that is 30% more dense than other embedded architectures as shown in Figure 9. This improved code density translates directly into better cache efficiency (e.g., since the executable code is smaller, more of the code can fit into the cache).

Microprocessors with on-chip instruction caches often have nondeterministic response to real-time events. Typically, it is impossible to know if the instruction cache contains the interrupt procedure for a newly detected interrupt request. The i960® RP processor solves this problem by allowing the programmer to permanently lock critical sections of code, such as interrupt handlers, in the instruction cache. The result is a faster and more deterministic interrupt response time.

The i960 RP processor provides a 2-Kbyte direct-mapped data cache. The inclusion of a data cache is a significant performance enhancement for applications that keep most of their data in RAM. A data cache also helps bus-intensive multimaster applications such as internetworking. While the processor accesses data in the internal caches, the memory bus is free for use by external agents such as Ethernet and Token Ring controllers.

The on-chip data RAM is a unique feature of the processor. A 1-Kbyte region of zero wait-state memory provides storage for data variables and interrupt vectors. Unlike a data cache, variables can never be “kicked out” of the data RAM. The on-chip data RAM provides high speed and deterministic access to data, which profiling compilers can take advantage of for critical variable storage.

Enhanced Bus Control Unit

The i960 RP processor bus control unit (BCU) integrates the functionality to reduce system cost and to support on-chip data caches. Integration of data caches introduced a new set of issues for software engineers. Data variable locality, code locality, and system architecture are now highly important.

The bus control unit addresses these design considerations by separating the external bus into physical and logical attributes.

Physical considerations for external buses provide the flexibility that system designers need. The i960 RP microprocessor supports mixed external bus widths. Execution of code can occur on 32-bit, 16-bit, or 8-bit external memory. For example, system initialization can be executed from a low-cost 8-bit wide ROM. The software executing from the 8-bit ROM transfers execution to the "Real" executable code located in a wider memory for higher performance.

Placement of peripheral components on the external bus can occasionally cause confusion within a system. If a system were to place an 8-bit peripheral on a 32-bit wide external bus, the software would be responsible for the proper addressing (every fourth byte) to access multiple locations in the peripheral. The i960 RP processor solves this problem by simply defining that portion of the external memory as 8-bit wide memory. The BCU will automatically address the peripherals transparent to application software. The i960 RP processor bus control unit allows the system designer to define eight independent regions within the address space of the i960 RP processor. Each region is 512-Mbytes in length with 8-bit, 16-bit, or 32-bit wide attributes.

The BCU also applies logical attributes to the external bus. Logical attributes refer to the data variables cached in the internal data cache. For example, application code typically has specific requirements for the data variables. Depending on the application, external bus masters often update variables with new data. When this occurs, an application would not want all of the data variables stored in the on-chip data cache.

The BCU accounts for the data caching by allowing the application to create logical templates. Each of the logical templates (LMCONs) defines regions within the 4-Gbyte address space with the attribute of "cacheable" or "non-cacheable." An LMCON describes the data access addresses allowed to be "cached" in the data cache. The i960 RP processor provides two logical templates. The templates consist of a base address register and a mask register. These register pairs provide the programmer with the capability to define a block of addresses as small as 4-Kbytes or as large as 4-Gbytes to be "cacheable" or "non-cacheable." Figure 10 shows an example of the BCU register programming.

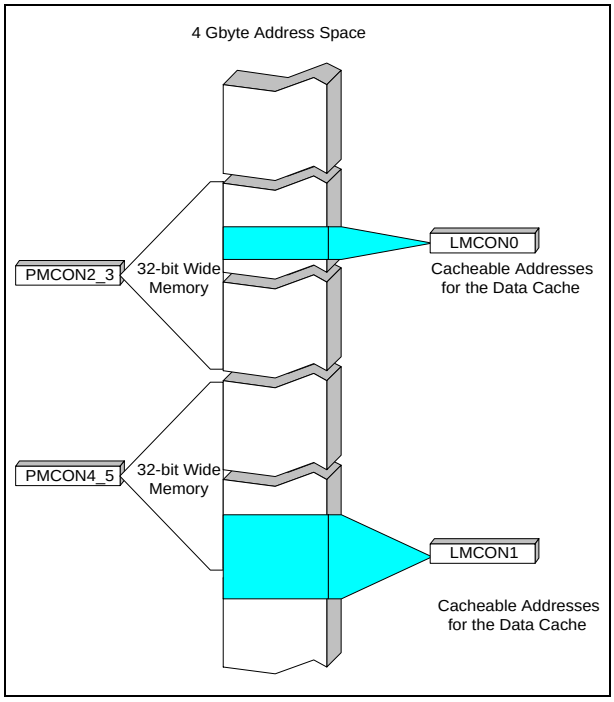

Figure 10. Physical and Logical Memory Control Example

The i960 RP processor supports homogeneous big-endian or little-endian data types. Hardware integrated into the BCU automatically converts the data for the different endian types. There is also transparent support for unaligned Big-endian and Little-endian data accesses.

Superior Interrupt Performance

Interrupt performance within embedded systems is critical. Many asynchronous events require immediate response in real time. However, the on-chip instruction caches can often create nondeterministic response times. The i960 RP processor improves the interrupt latency and throughput issues.

The interrupt controller supports two operational modes: dedicated and expanded mode. Dedicated mode requires the external device to signal an interrupt request to the processor with a dedicated signal toggling. When the interrupt controller detects an active signal, it determines the priority and vector of the associated interrupt. Dedicated mode supports either edge or level detection and fast or debounced sampling. Expanded mode requires the external device to place an interrupt vector on the interrupt pins, requiring the interrupt controller to read the pins and determine the priority and vector of the interrupt.

The processor core determines the address of the first instruction of the interrupt service routine from the priority vector. The i960 RP processor allows for caching of the interrupt vector addresses that point to the addresses of the first executable instruction. Reserved for this purpose are the first 64 bytes of on-chip data RAM. Caching these vectors reduces the external bus traffic, providing higher throughput.

To reduce the time necessary for the processor to fetch the first executed instruction, the i960 RP processor can permanently lock critical sections of code, such as interrupt handlers, in the instruction cache. The result is faster and more deterministic interrupt response time.

The core performs a task switch to the interrupt service routine. One feature often overlooked in the i960 architecture is that the processor performs the overhead associated with a task switch. This includes allocating a fresh register set, switching to a dedicated stack, and saving the previous internal state for later resumption of the interrupted task. The i960 RP processor can reduce the task switch time by reserving local register set(s) for interrupts priority 28 and higher. Reserving these register sets is another method of ensuring deterministic interrupt response.

These improvements enable the processor to ensure low interrupt latency for high-performance interrupts.

i960® RP Processor Integrated Peripherals

The i960 RP processor core integrates two on-chip peripheral's, an integrated timer unit and an integrated interrupt controller unit. Both units provide flexibility for many applications.

The integrated timer unit provides two identical 32-bit timers. The timers have a single-shot mode and an auto-reload mode for continuous operation. Each timer provides an independent interrupt request to the i960 RP microprocessor interrupt controller. The timer registers interface through internal memory-mapped addresses. When enabled, the timer control circuitry generates a fault when detecting unauthorized writes from user mode.

The interrupt controller unit (ICU) provides a flexible low-latency means for requesting interrupts. This unit handles the posting of interrupts requested by hardware and software sources. The interrupt controller, acting independently from the core, compares the priorities of posted interrupts with the current process priority, off-loading this task from the core. The interrupt controller provides the following features for handling hardware-requested interrupts:

- Support of up to 240 external sources
- Eight external interrupt pins
- One non-maskable interrupt pin
- Two internal timer sources for detection of hardware-requested interrupts
- Edge or level detection on external interrupt pins
- Debounce option on external interrupt pins

The application program communicates to the interrupt controller with six memory-mapped control registers. The interrupt control register and interrupt map control registers provide configuration information. The interrupt pending register posts hardware-requested interrupts. The interrupt mask register enables the application to selectively mask hardware-requested interrupts.

i960® RP PROCESSOR DATA FLOW REQUIREMENTS

I/O processors control data movement between peripheral components, local and system memories. The data packets can range from small packets, such as control descriptors, to large amounts of data in the megabyte range.

Figure 11 summarizes the three basic data flows. They include:

- Bridging of data from one PCI bus to the other PCI bus
- Direct data transfers between the primary or secondary PCI bus and the local bus
- Indirect data transfers between primary or secondary PCI buses and the local bus

The optimized i960 RP architecture provides high data transfer throughput between the PCI buses and the local bus. The PCI-to-PCI bridge, address translation units (ATUs) and the DMA channels contain deep 64-byte queues to reduce latency and provide high throughput.

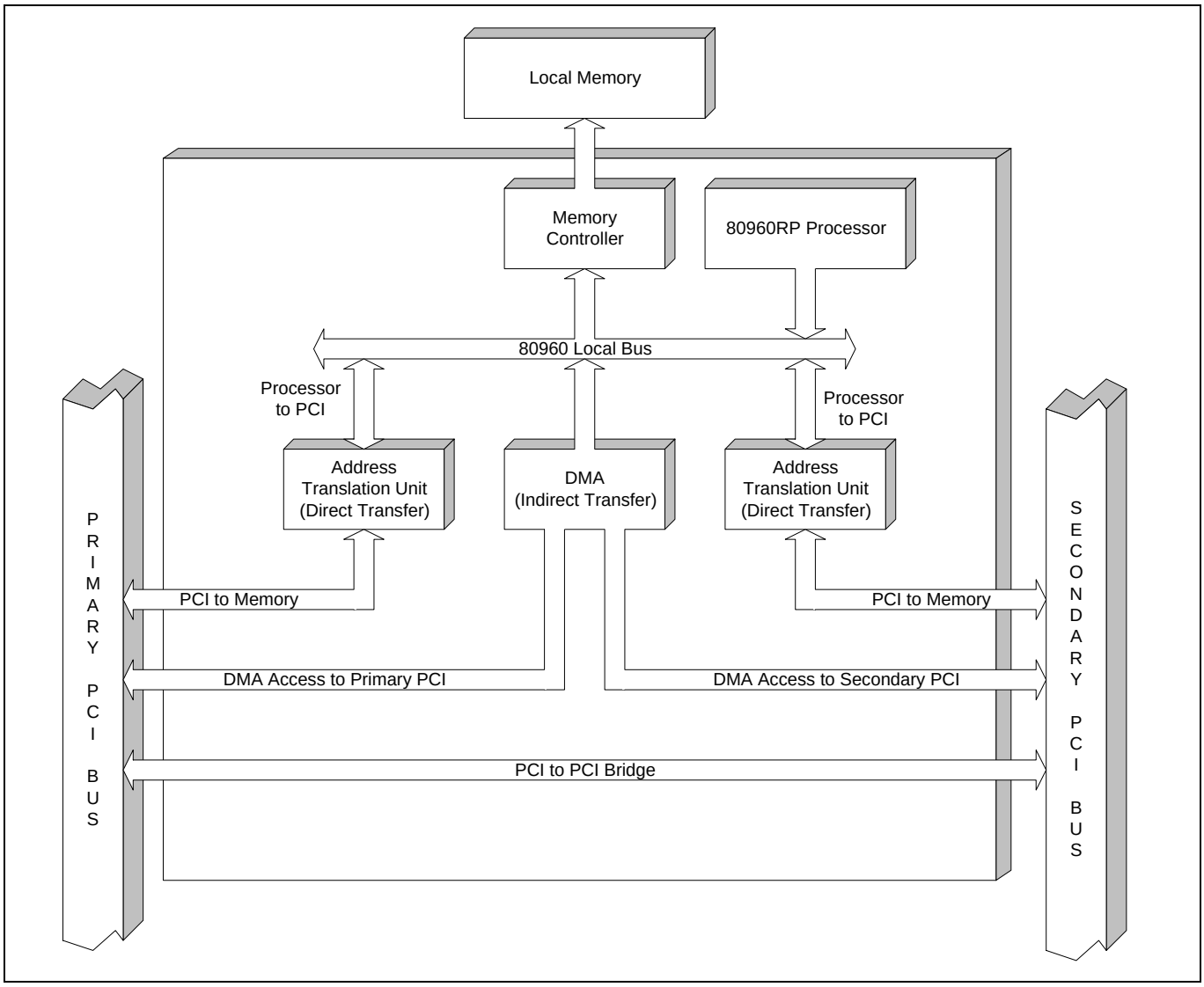

Figure 11. i960® RP Processor Data Flow Requirements

PCI-TO-PCI BRIDGE

The primary function of the PCI-to-PCI bridge integrated in the i960® RP processor is to create an electrically isolated PCI bus. This enables the system designer to connect I/O components directly to the PCI bus and to add additional PCI card slot connectors. However, the bridge incorporates many features that improve overall system-level performance by reducing bus traffic. In addition, bridge design features, such as VGA and ISA addressing support, provide the system designer with the greatest flexibility in creating a complete PCI system.

Electrically Isolated PCI Buses

The PCI-to-PCI bridge unit extends the limitations of 10 electrical PCI loads at frequencies of 33 MHz. The PCI-to-PCI bridge unit enables additional I/O components or PCI card slots within the system by creating an electrically isolated secondary PCI bus.

The 80960RP bridge uses the concept of hierarchical buses where each bus in the hierarchy is electrically a separate entity, but are logically one bus. The system BIOS may assign bus numbers to each PCI bus located within the system. The host processor programs the bus numbers in the hierarchy into the bridge.

The PCI interfaces on the 80960RP processor comply with 5.0V PCI electrical requirements. When connected to a PCI system, the secondary interface of the bridge is the first electrical load on the isolated PCI bus. Up to nine additional loads can be connected to the secondary PCI bus at 33 MHz. Additional i960 RP processors downline (connected to the secondary PCI bus interface) let the designers extend the hierarchies of buses.

Isolated Data Flow

The bridge design isolates unnecessary data traffic between the primary PCI bus and the secondary PCI bus. The PCI data traffic falls into three unique address spaces:

- 4 Gbyte memory address space
- 64 Kbyte I/O address space
- RP processor bridge configuration space

The 80960RP processor bridge supports the decoding of these three different address spaces. The bridge configuration space contains the internal registers for defining the different address ranges supported by the bridge. The bridge isolates PCI transactions initiated on and intended for the same PCI bus, thus reducing the traffic on the other PCI bus. This is also known as an opaque PCI-to-PCI bridge.

i960® RP Processor Bridge Address Decode

All PCI-compliant devices and cards are dynamically assigned memory and I/O addresses at powerup. Each device or card responds to these programmed values when accessed on the PCI bus. The host CPU allocates memory and interrupts to all devices and cards found on the secondary PCI bus. In addition, the host CPU programs the bridge with the respective address range that the bridge decodes, and performs data forwarding between the primary and secondary PCI buses.

The values programmed into the PCI-to-PCI bridge define the contiguous range (or window) of addresses. This window of programmed addresses create the memory and I/O address spaces for the secondary PCI address space. *Positive decode* means the bridge forwards any address on the primary side of the bridge that falls within the programmed secondary space to the secondary side. The bridge ignores any address presented on the primary PCI interface that falls outside the secondary address space. The bridge isolates the transaction from the secondary PCI bus. The primary PCI interface performs this *positive decode*. Figure 12 shows the addresses (once programmed by the host CPU) forwarded between the PCI buses.

The positive decode mechanism applies to the primary interface of the PCI bridge. The secondary interface of the bridge allows the system designer to specify any one of the three modes of operation. They are:

- Inverse Positive Decoding
- Subtractive Decoding
- Positive Decoding

The secondary PCI interface of the bridge works in reverse of the primary side. *Inverse positive decode* means the bridge ignores any addresses within the programmed secondary address space. Thus, the bridge isolates the transaction from the primary PCI bus. The bridge forwards the addresses that fall

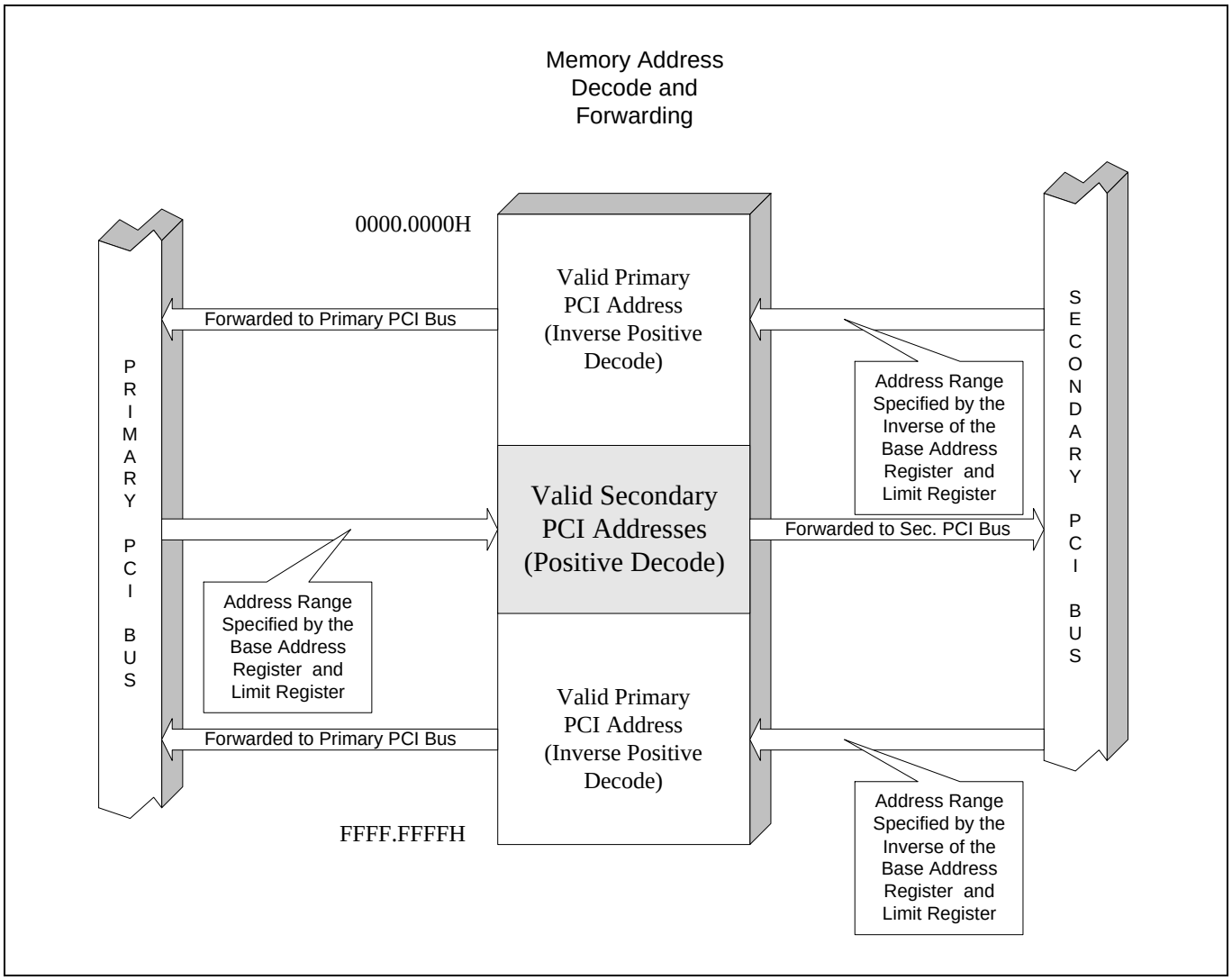

Figure 12. i960® RP Processor Bridging Operation

outside the secondary PCI address space from the secondary PCI bus to the primary PCI bus.

The secondary PCI interface of the bridge also contains an address forwarding mechanism that does not require decoding of the addresses on the secondary PCI bus. *Subtractive Decode* requires the secondary interface to claim the transaction on the fifth clock (assert **S_DEVSEL#**) after detecting **S_FRAME#** asserted by an initiator on the secondary bus. By PCI definition, a secondary bus target will claim any transaction initiated on the secondary bus by asserting the **S_DEVSEL#** prior to the fifth clock. Thus, *Subtractive Decode* will forward only the unclaimed transactions to the primary PCI bus.

Peer-to-Peer data transfers require blocks of addresses dedicated to those devices. The bridge provides an additional *positive decode* mechanism on the secondary interface, independent of the *subtractive decode mechanism*. This mode allows secondary PCI devices/cards to transfer data between the PCI devices/cards without requiring primary PCI bus address space.

Each of the memory address decode regions also contain attributes to determine the rules for the transaction. For example, the memory address space may be prefetchable or non-prefetchable. If the transaction is within a prefetchable memory address region, the bridge may read additional data to anticipate the data transfer from the initiator, resulting in higher throughput for each transaction.

ISA Address Forwarding

The i960 RP processor bridge implements an ISA mode that enables the bridge address decode to be ISA aware. ISA mode only affects I/O addresses within the address range defined by the I/O address registers. This mode supports downstream ISA I/O cards connected to subordinate PCI buses via a PCI-to-ISA bridge.

When enabled, the bridge filters out and does not forward I/O transactions with addresses in the upper 768 bytes (300H) of each naturally aligned 1 Kbyte block. Conversely, I/O transactions on the secondary bus will inversely decode the ISA addresses and forward I/O transactions with addresses in the upper 768 bytes of each naturally aligned 1 Kbyte block.

ISA cards only decode the lower 10 bits of an address (1 Kbyte). General I/O assigns the upper 768 bytes of the 1 Kbyte block. Because these cards do not decode the upper 6 bits of the 16-bit I/O address, the ISA address is aliased 64 times in the 64 Kbyte I/O address space. The combination of ISA addressing modes and the 4 Kbyte I/O address granularity results in an address decode that is similar to EISA slot decoding. Devices on the secondary interface can use the first 256 bytes of each 1 Kbyte block. ISA addressing does not affect ordering, posting or error handling behavior of the 80960RP processor PCI-to-PCI bridge. Figure 13 shows which I/O addresses that are forwarded upstream and downstream during ISA mode.

VGA Addressing and Snooping Support

VGA devices on the secondary PCI bus require special decode logic which is designed into the 80960RP processor bridge. This logic provides system designers with greater flexibility when designing the system.

The issues related to VGA-compatible devices with the i960 RP processor bridge include VGA-ISA compatible addressing and VGA palette snooping. To support a VGA device on a downstream bus from the 80960RP processor, the bridge can recognize and forward VGA addresses on the primary interface to the secondary interface. The bridge unit also supports downstream graphics devices that have to snoop VGA palette accesses on the primary bus.

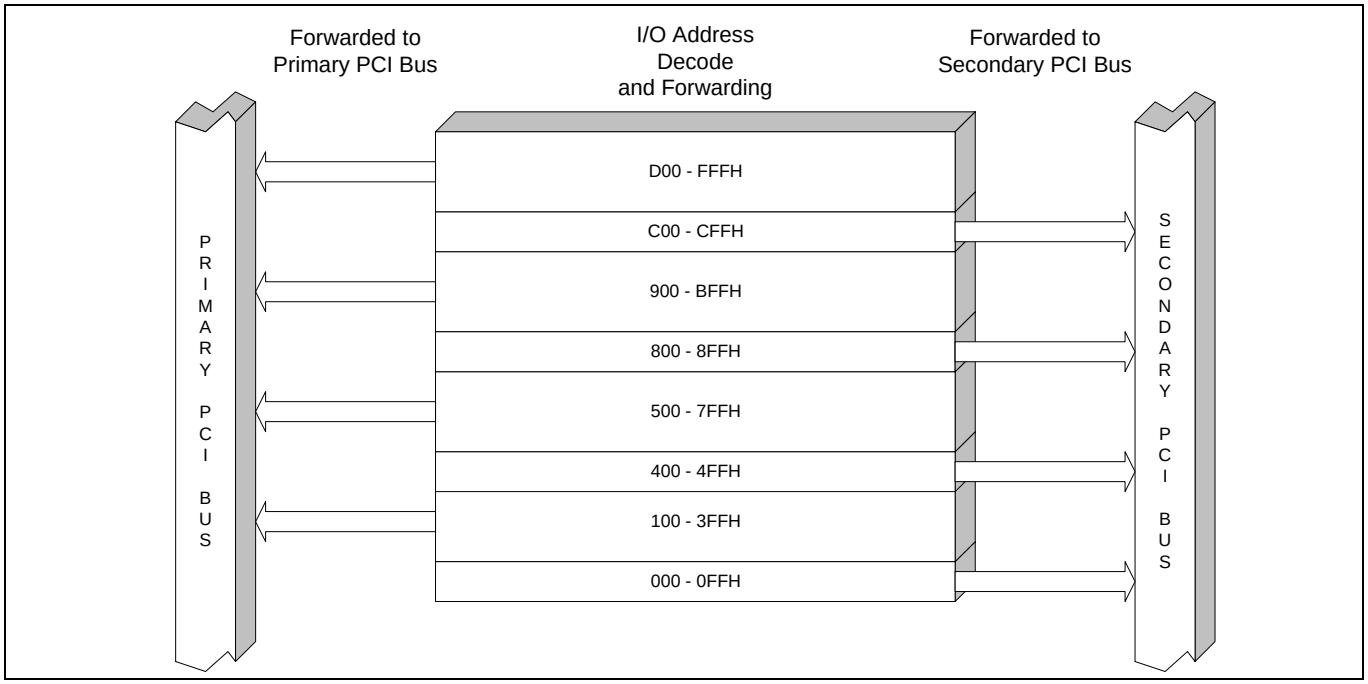

Figure 13. ISA Mode Address Forwarding

VGA-Compatible Addressing

Compatibility with VGA addressing allows the 80960RP processor to support VGA frame buffer addressing and VGA register addressing. When enabled, the bridge will positively decode memory accesses to a VGA frame buffer and I/O accesses to VGA registers on a secondary bus. The primary interface positively decodes the following addresses:

- VGA memory accesses: addresses 000A.0000H — 000B.FFFFH
- VGA I/O accesses: addresses 3B0H — 3BBH and 3C0H — 3DFH

VGA-compatible addressing is not dependent on the address ranges programmed into the bridge configuration registers as described by the bridge address decoding section. The bridge forwards VGA addresses from the primary bus to the secondary bus and blocks VGA address from secondary to primary regardless of the defined address ranges. In addition, VGA compatible addressing is not dependent on the ISA addressing mode selected. Figure 14 summarizes the VGA-compatible addressing.

64-Bit Addressing with Dual Address Cycle

The i960 RP processor supports the 64-bit address extension defined in the PCI local bus extension. 64-bit addressing takes the form of the PCI dual address cycle (DAC). A DAC cycle consists of breaking the 64-bit address into two 32-bit address cycles. The first address cycle contains the lower 32-bits of address with the byte enable/command pins (**BE[3:0]#**) set to specify a DAC cycle. The second address cycle contains the upper half of the 64-bit address with the byte enable/command pins specifying the cycle type.

The bridge unit provides support for upstream DAC cycles. The secondary interface of the 80960RP bridge performs subtractive decoding to support DAC cycle forwarding. If no other agent claims the transaction within the low, medium or high decode times, the bridge will claim the transaction and forward the transaction upstream to the primary PCI bus. The bridge ignores all DAC transactions on the primary interface of the bridge.

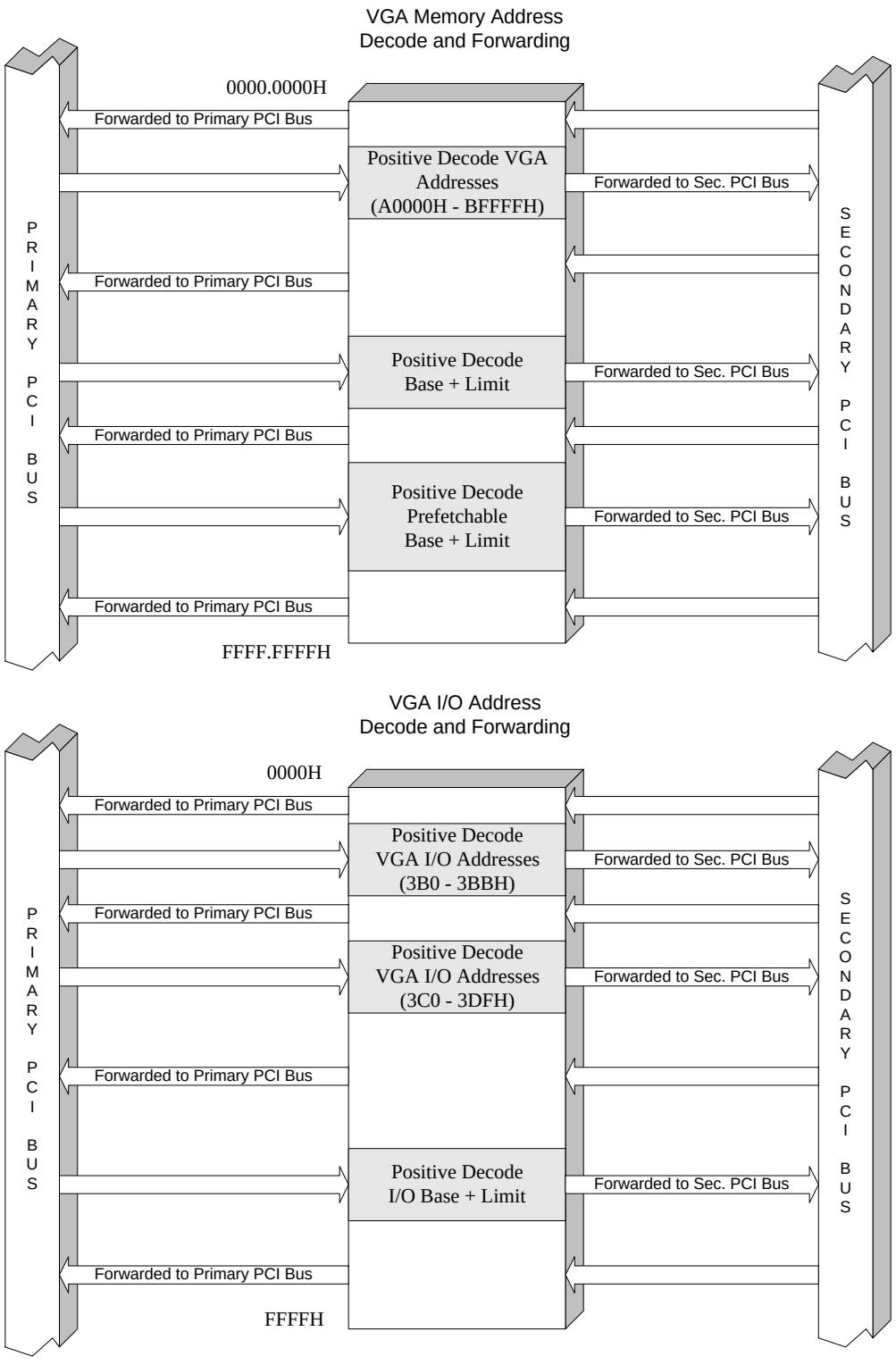

Figure 14. VGA Compatible Address Forwarding

Bridge Queues

The i960 RP processor contains deep queues within the PCI-to-PCI bridge. These queues improve the system throughput by allowing transactions to complete on the initiating bus prior to completing on the target bus.

The bridge queues support high-performance bandwidth on both PCI buses. The bridge queue implementation provides a FIFO-style architecture that allows for data streaming to and from the queue simultaneously. This data streaming, for PCI write transactions, allows the bridge to transfer data up to 132 Mbytes/sec rate.

The bridge unit contains six dedicated queues to support both downstream (primary to secondary) and upstream (secondary to primary) transactions. These queues include:

- 64-byte downstream data queue
- One 32-bit downstream write cycle address queue
- One 32-bit downstream read cycle address queue
- 64-byte upstream data queue
- One 32-bit upstream write cycle address queue
- One 32-bit upstream read cycle address queue

The architecture allows the bridge to simultaneously support transactions occurring on each PCI bus. This implementation increases the system throughput by allowing the bridge to claim and complete the transactions. The architecture does not stall the initiating PCI bus while waiting to acquire the target PCI bus.

PCI Bridge Transactions

The bridge unit supports both delayed PCI transactions and posted PCI transactions. These transaction types are dependent on the cycle type generated by the PCI master when the bridge claims the transaction.

Delayed transactions improve the overall bus efficiency for target PCI devices that require high initial latency. For example, when accessing a PCI-to-PCI bridge, the target PCI bus activity prevents a transaction from completing. The bridge processes all transactions as delayed transactions, except for *memory write* and *memory write and invalidate* cycle types. The system designer controls these two transaction types through configuration registers as either delayed transactions or posted transactions.

A delayed transaction consists of two parts: a request phase and a completion phase. In the request phase, the bridge latches the address, command, byte enables, and data (for write transactions only) information required to complete the transaction. The bridge immediately signals a **retry** to the initiator, freeing the PCI bus. The bridge performs the request on the target bus on behalf of the initiator. During read cycles the bridge stores the returning data and the target response in the bridge queues. During write cycles the bridge records the target response. The completion phase waits for the original initiator to repeat the original request in order to pass back the data (in the case of read cycles) and return the target response to complete the transaction.

The bridge unit supports posted transactions for PCI *memory write* or *memory write and invalidate* cycle types. For posted transactions, the bridge latches the address, command, byte enable, and data required to complete the transaction. The bridge signals the termination (i.e., **normal termination** or **disconnect**) to the initiator upon filling the bridge queues or when the initiator does not have any additional data to transfer. This normal termination notifies the initiator that it does not have to repeat the transaction. The bridge initiates the transaction on the target bus upon detecting the ownership of the bus.

Posted transactions do not require the initiator to repeat the request, thus providing optimized PCI bus utilization and releasing the initiator to continue with other PCI bus operations.

The bridge supports delayed transactions in all directions for the following cycle types:

- *Memory Read*
- *Memory Read Line*
- *Memory Read Multiple*
- *I/O Read*
- *Configuration Read*
- *Configuration Write*
- *I/O Write*
- *Memory Write* (configurable for delayed transactions or posted transactions)
- *Memory Write & Invalidate* (configurable for delayed transactions or posted transactions)

PCI Data Streaming

During PCI write transactions the bridge unit latches the data as it enters the bridge. Upon receiving the address and the first data, the bridge asserts the request for the target bus in an attempt to acquire the target bus prior to the filling of the bridge queues. Upon acquiring the target bus while the initiator sends data to the bridge, the bridge streams the data from one PCI bus to the other without breaking the burst transaction into multiple transactions.

PCI read transactions behave similarly. The bridge unit latches the information from the initiator and performs the transaction on the target bus. The bridge reads the data from the target bus and stores the data in the queue. The amount of data read from the target depends on the memory read cycle type and the cache line size value. For the *memory read multiple* command, the bridge fills the queue. When the initiator repeats the original request and the bridge detects an exact match for the read transaction, the bridge returns the data to the initiator. Upon draining the first data value, the bridge requests the target bus in an effort to stream data across the bridge. If successful in acquiring the target bus, the bridge can sustain the full 132 Mbyte/sec bandwidth.

ADDRESS TRANSLATION UNITS (ATU)

The i960® RP processor provides an interface between both PCI buses and the 132 Mbyte/sec processor's local bus. This interface consists of two Address Translation Units (ATUs) and a messaging unit. The ATUs support both inbound and outbound address translations. The first address translation unit, called the primary ATU, provides direct access between the primary PCI bus and the 80960 local bus. The second address translation unit, called the secondary ATU, provides direct access between the secondary PCI bus and 80960 local bus.

The primary ATU allows PCI masters on the primary PCI bus to initiate transaction to the 80960 local bus and allows the 80960RP processor to initiate transactions to the primary PCI bus. The secondary ATU performs the same function on the secondary PCI bus and for secondary PCI bus masters. *Inbound transactions* are initiated on the PCI bus and targeted to the 80960 local bus*. Outbound transactions* initiated on the 80960 local bus are targeted for the PCI bus.

Inbound address translation maps PCI memory space into the memory space. This allows PCI bus masters to directly read or write data into the memory. In addition, the ATUs support inbound burst data transfers from a PCI bus master up to 2 Kbytes per transaction.

The ATUs implement an address windowing scheme to determine which addresses to claim and translate to the appropriate bus. The address windowing mechanism for inbound translation involves the following steps:

- Address detection — decode the 32-bit PCI address and determine if the address falls within the address window defined for the inbound ATU by the base address register and the limit register.
- Transaction claiming — claim the PCI transaction by asserting **P_DEVSEL#** or **S_DEVSEL#.**
- Address translation — translate the 32-bit PCI address to a 32-bit 80960 local bus address.

Figure 15 shows an example of inbound address translation.

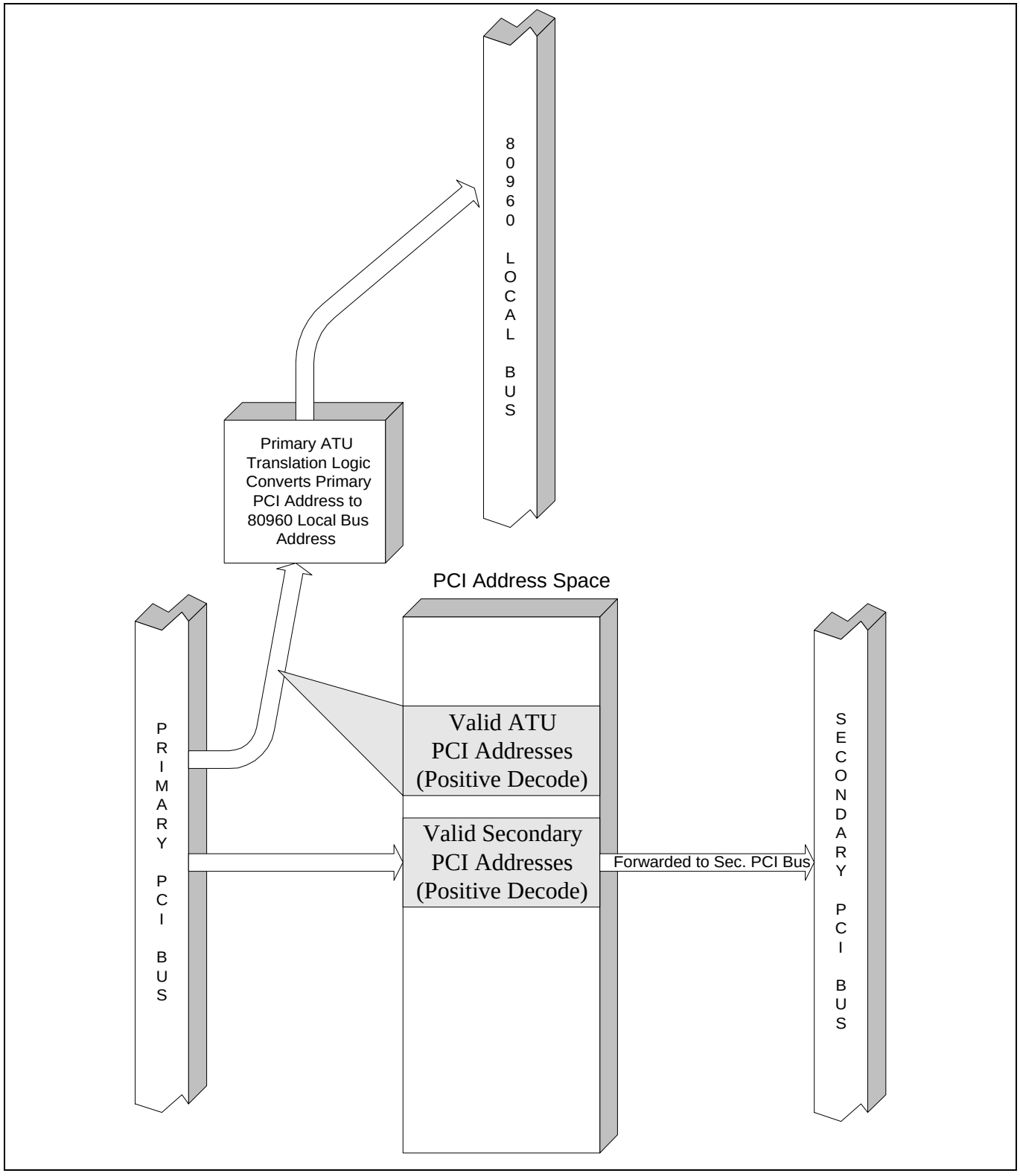

Figure 15. Primary ATU Inbound PCI Address Translation Example

ATU Queues

The i960 RP processor contains deep queues within each ATU. These queues improve the system throughput by allowing the transaction to complete on the initiating bus prior to completion on the target bus.

The ATU queues support high-performance bandwidth on both the PCI bus and the 80960 local bus. The ATU queue implementation provides a FIFO-style architecture allowing for simultaneous data streaming to and from the queue. For PCI write transactions, data streaming allows the ATU to transfer data at the 132 Mbyte/sec rate.

The ATU contains six dedicated queues to support both inbound (PCI to 80960 local bus) and outbound (80960RP processor to PCI) transactions. These queues include:

- 64-byte inbound data queue
- 32-bit inbound write cycle address queue
- 32-bit inbound read cycle address queue
- 64-byte outbound data queue
- 32-bit outbound write cycle address queue
- 32-bit outbound read cycle address queue

The architecture allows the ATUs to simultaneously support transactions occurring on both buses. This implementation increases system throughput by allowing the ATU to claim and complete the transactions. Similarly to the bridge, this architecture will not stall the initiating PCI bus while waiting to acquire the target 80960 local bus.

ATU Inbound Transactions

The ATUs supports delayed and posted inbound PCI transactions. The transaction type depends on the cycle type generated by the PCI master when the ATU claims the transaction.

Delayed transactions improve the overall bus efficiency for target PCI devices that require high initial latency. The ATUs process all transactions as delayed transactions except PCI *memory write* cycle types. The ATUs process all PCI *memory write* cycle types as posted transactions. The delayed and posted transaction types operate the same as the bridge transactions.

Inbound ATU Data Streaming

During PCI write transactions the queues latches the data as it enters the ATU. Upon receiving the address and the first data, the ATU asserts the request for the local bus in an attempt to acquire the local bus prior to filling the ATU queues. Upon acquiring the local bus while the initiator sends data to the ATU, the ATU streams the data from the PCI bus to the local bus without breaking the transaction into multiple transactions.

PCI read transactions behave similarly. The ATUs latch the information from the initiator and perform the transaction on the local bus. The ATU reads the data from the local bus and stores the data in the queue. The amount of data read from the target depends on the behavior of the **P_FRAME#** or **S_FRAME#** signals. This signal provides a hint to the ATU for prefetching data. If the master deasserts the **P_FRAME#** or **S_FRAME#** signal on the clock after the address cycle, the initiator is requesting a single data value. Otherwise, the ATU fills the queue. When the initiator repeating the original request and the ATU detecting an exact match for the read transaction, the ATU returns the data to the initiator. Upon draining the first data value, the ATU requests the local bus in an effort to stream data to the PCI bus. If successful in acquiring the local bus, the ATUs can sustain the full 132 Mbyte/sec bandwidth on both the PCI bus and the 80960 local bus.

Outbound ATU Transactions

Outbound ATU transactions enable the 80960RP processor to access the PCI buses directly. These transactions perform address translation from the 80960 local bus to the PCI bus. Since the 80960RP processor only supports memory transactions, blocks of 80960 local bus addresses translate the 80960RP read and write commands into various PCI cycle types. These include:

- PCI memory read and write cycles
- PCI I/O read and write cycles
- PCI Dual Address Cycle (DAC) read and write cycles
- PCI configuration read and write cycles

The 80960 local bus reserves three windows for each ATU. The memory windows generate PCI memory cycles (PCI *memory read* or PCI *memory write*) when accessed by the 80960RP processor.

These windows translate the 80960 local bus address into a PCI address. If the 80960RP processor performs a read cycle to an address in the window, the ATU translates the local bus address into a PCI address, generates the PCI read cycle, and returns the data to the 80960RP processor. When the 80960RP processor writes to the memory window, the ATU latches the data (a maximum of 16 bytes), translates the local bus address to the PCI address, and generates a PCI write cycle followed by the data.

One local bus address window generates PCI I/O read cycles and PCI I/O write cycles. The ATU translation operates similarly to the PCI memory windows.

The third window generates a PCI DAC cycle on the PCI bus. A preprogrammed register contains the value used to generate the upper portion of the 64-bit PCI address. The ATU translates the 80960 local bus address to form the lower portion of the PCI address.

During any PCI read cycle the ATU places the 80960RP processor in a back-off mode of operation. This frees the 80960 local bus for another local bus master while the PCI read transaction completes.

ATU Direct Addressing Transactions

The outbound ATU direct addressing enables the 80960RP processor to directly access the PCI buses without any address translation. These transactions take the 80960RP address directly from the 80960 local bus to the PCI bus.

The outbound ATU direct addressing allocates one window of addresses on the 80960 local bus. The memory windows generate PCI memory cycles (PCI *memory read* or PCI *memory write*) when accessed by the 80960RP processor. This window behaves similarly to the outbound address translation with the exception of the PCI address.

The 80960RP processor controls the direct addressing mode via control bit in the ATU configuration registers. The programmability allows the direct addressing to generate either primary PCI transactions, secondary PCI transactions, or be disabled. The direct addressing window uses the lower 2 Gbytes of 80960 local bus addresses.

Generating PCI Configuration Cycles

The outbound ATUs provide a port programming model for generating outbound PCI configuration cycles. Performing an outbound configuration cycle to either the primary or secondary PCI bus involves up to two 80960 local bus cycles to two hardware registers for each ATU.

The 80960RP processor must write to an outbound configuration address register (primary or secondary) the PCI address used during the configuration cycle. 80960RP read or write cycles to the outbound configuration data register (primary or secondary) initiates the PCI transaction. A 80960RP read to the outbound configuration data register generates a PCI configuration read cycle type on the PCI bus. The ATU obtains the address from the outbound configuration address register. The ATU returns the PCI data to the outbound configuration data register. A 80960RP write to the outbound configuration data register generates a PCI configuration write cycle type on the PCI bus. The ATU obtains the address from the outbound configuration address register and data from the outbound configuration data register.

Configuration cycles are non-burst and restricted to a single PCI DWORD cycle.

PRIVATE PCI DEVICES

An add-in card application using an i960® RP processor may require PCI devices to be hidden from the host processor(s). These PCI devices require special attention when the system designer connects the **IDSEL#** signals to the 80960RP secondary PCI bus. Hidden PCI devices, otherwise known as private PCI devices, must not be visible to the host processor when the BIOS software polls for the PCI devices/cards connected to the hierarchy of PCI buses. Otherwise, the BIOS will allocate PCI address space for these devices.

The i960 RP processor requires total control of the private PCI devices to implement an intelligent I/O subsystem. From the host processor, the intelligent I/O subsystem appears as a single PCI unit. In reality, multiple PCI devices exist under the control of the 80960RP processor.

The i960 RP processor supports up to 10 private PCI devices on the secondary PCI bus. In order to support private PCI devices, custom hardware functions in the bridge and RP ATUs provide the functionality to implement an intelligent I/O subsystem.

Connecting the PCI devices **IDSEL#** signal to any of the **S_AD[15:11]** pins creates private PCI devices. The bridge hides an additional five PCI devices on the secondary PCI bus by a custom hardware mechanism built into the bridge. This mechanism blocks the assertion of the **S_AD[20:16]** signals during configuration cycles generated by the host processor. A control register in the bridge extended configuration space provides independent control of signal blocking during the configuration cycles.

The outbound ATUs perform PCI configuration to the private PCI devices. The **S_AD[20:11]** signals are never blocked from an outbound ATU configuration cycle, thus allowing the 80960RP processor to access and control all PCI devices on the secondary PCI bus.

Private PCI Address Space

The private PCI devices on the secondary PCI bus require PCI address space. Special hardware designed into the ATUs and bridge enables the private PCI addresses. The 80960RP processor uses the private PCI address space to communicate with the private PCI devices.

The bridge unit, which normally performs inverse positive decoding on the secondary PCI bus, contains registers that positively decode addresses and do not forward them to the primary PCI bus. These private PCI addresses enable peer-to-peer PCI transactions. For example, there may be two intelligent I/O subsystems on the secondary PCI bus. Each subsystem can contain the intelligence to transfer data from one subsystem directly to the second subsystem without requiring any bandwidth from the primary PCI bus.

The secondary ATU provides the second form of private PCI addresses on the secondary PCI bus. The inbound ATU functions on the secondary PCI bus claim the transactions and notify the bridge. Therefore, the bridge does not forward the transaction to the primary PCI bus. Figure 16 shows an example of both private PCI address spaces.

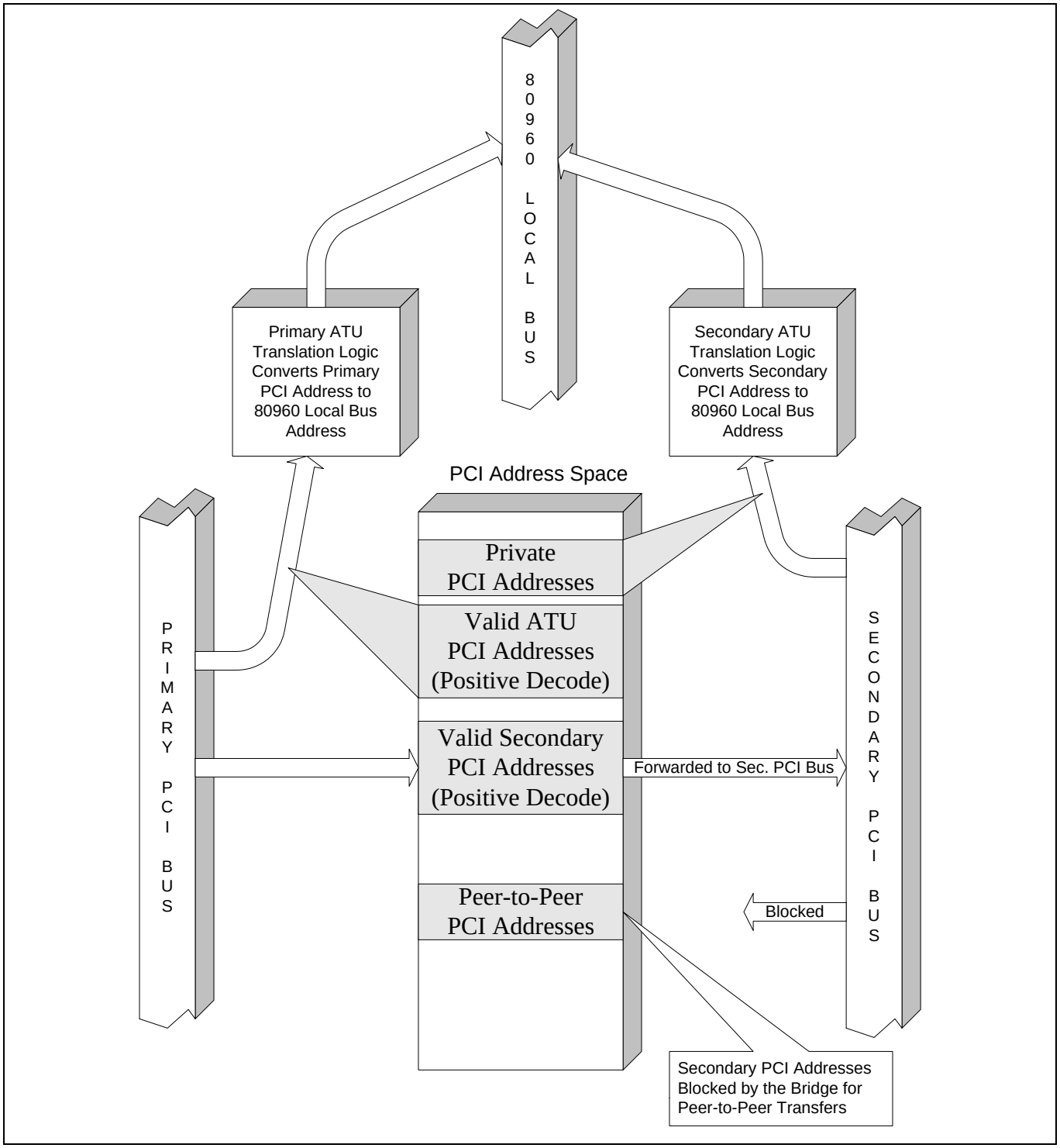

Figure 16. Private PCI Address Space

MESSAGING UNIT

The Messaging Unit (MU) provides a mechanism to transfer data between the PCI system and the 80960RP processor and to notify the respective system of the arrival of new data through an interrupt.

The MU has four distinct messaging mechanisms. Each allows a host processor or external PCI agent and the 80960RP processor to communicate through message passing and interrupt generation. The four mechanisms are:

- Message registers
- Doorbell registers
- Circular queues
- Index registers

Message Registers

The i960® RP processor sends and receives messages via the special message registers. When written, the message registers may generate an interrupt to either the 80960RP processor or the PCI interrupt signals. The host processor sends inbound messages for the 80960RP processor and the 80960RP processor sends outbound messages for the host processor.

When the 80960RP processor writes to an outbound message register, the message unit may generate an interrupt on the **P_INTA#**, **P_INTB#**, **P_INTC#**, or **P_INTD#** interrupt pins. The ATU interrupt pin register determines which interrupt line the message unit generates the interrupt. When an external PCI agent writes to an inbound message register, the message unit may generate an interrupt to the 80960RP processor.

Doorbell Registers

There are two doorbell registers: the Inbound Doorbell Register and the Outbound Doorbell Register. The Inbound Doorbell Register allows external PCI agents to generate interrupts to the 80960RP processor. The Outbound Doorbell Register lets the 80960RP processor generate PCI interrupts. Both Doorbell Registers hold a combination of hardware and software generated interrupts. They contain interrupt status from other Messaging Unit mechanisms, and they also allow software to set individual bits to cause an interrupt.

Circular Queues

The MU implements four circular queues. There are two inbound queues and two outbound queues. In this case, inbound and outbound refer to the direction of the flow of messages. The data passed through the queue are either free messages or posted messages. Posted messages contain data instructing the destination processor to perform operations. Free messages contain data notifying the destination processor the data operation completed and the message is available for reuse.

The four Circular Queues pass messages in the following manner. The MU designates one of the inbound queues for free messages and one queue for posted messages. Similarly, the MU designates one of the outbound queues for free messages and one queue for posted messages. Figure 17 shows an overview of the Circular Queue operation. External PCI agents access the circular queues through two port locations in the PCI address space. The external PCI agents directly read or write the inbound and outbound queue ports.

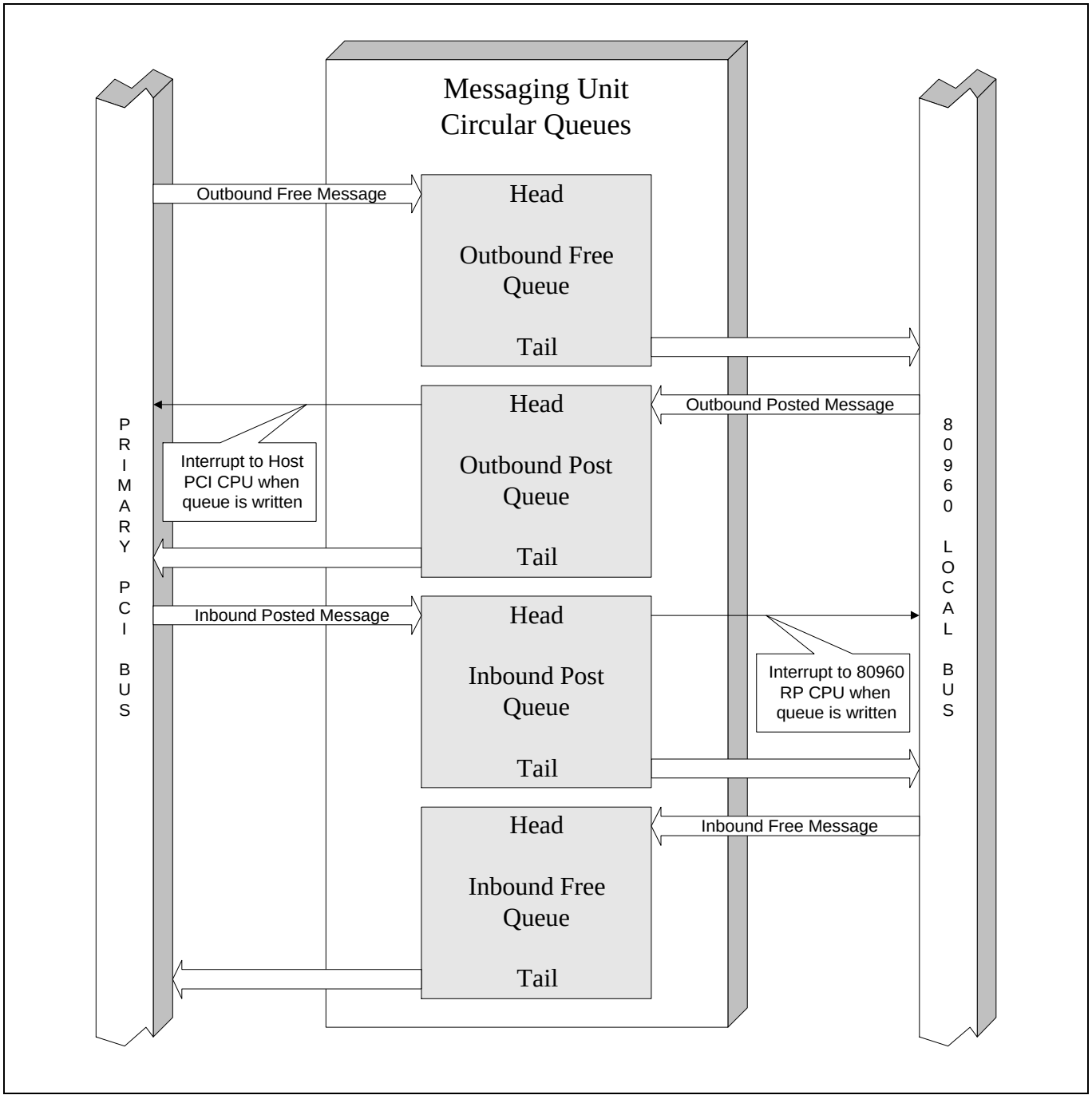

Figure 17. Overview of Circular Queue Operation

The i960 RP processor local memory provides for data storage in the circular queues. Each entry in the queue is a 32-bit data value. Each read from or write to the queue may access only one queue entry. The queue ports support single word transactions.

Each circular queue contains a head pointer and a tail pointer. Writes to a queue occur at the head of the queue and reads occur from the tail. Either the 80960RP processor or the MU hardware increments the head and tail pointers. The pointers contain offsets into the circular queue and range from zero to the circular queue size minus one. The pointers increment after the queue access. Both pointers wrap around to zero when they reach the circular queue size.

The Messaging Unit will generate an interrupt to the 80960RP processor or generate a PCI interrupt under certain conditions. When an external PCI agent writes to the post queue, the MU generates an interrupt to notify the receiver of a new posted message.

The size of each circular queue can range from 16 Kbytes (4096 words) to 256 Kbytes (65536 words). All four queues must be the same size and must be contiguous. Therefore the total amount of local memory needed by the circular queues ranges from 64 Kbytes to 1 Mbytes. A programmable MU configuration register determines the queue size.

Index Registers

The index registers are a set of 1004 registers that, when written to by an external PCI agent, can generate an interrupt to the 80960RP processor. These registers are for inbound messages only.

The i960 RP processor allocates storage for the index registers in local memory. The index address register contains the address of the access stored in the index register. This register stores the address of the index register written. This simplifies the 80960RP processor's effort to determine which register the external PCI agent accessed.

DMA CONTROLLER

The i960® RP processor contains an integrated hardware DMA controller. This DMA controller supplies three channels for performing high-throughput memory transfers between the PCI buses and the 80960 local memory. Channels 0 and 1 perform DMA transfers between the primary PCI bus and the 80960 local memory. Channel 2 transfers blocks of data between the secondary PCI bus and the 80960 local memory. All channels are identical, except for channel 0 which contains additional support for demand-mode transfers.

Each DMA channel uses direct addressing for both the PCI bus and the 80960 local bus. The channel supports data transfer to and from the full 64-bit address range of the PCI bus. Both the PCI bus interface and the 80960 local bus interface support zero wait-state data transfers, providing 132 Mbyte/sec throughput from bus to bus.

Each DMA channel supports simultaneous read and write transactions on the PCI bus and the 80960 local bus. This enables the DMA controller to perform data streaming when the DMA owns both the PCI bus and the 80960 local bus. The DMA controller supports PCI-style burst accesses on the 80960 local bus.

Hardware Unaligned DMA Transfers

Every channel of the DMA controller contains a hardware packing and unpacking unit. This unit allows the DMA controller to perform unaligned DMA transfers. The application programmer can program any possible combination of source and destination addresses with no performance degradation.

The packing and unpacking unit performs the data movement and prepares the data for the correct byte lanes on the respective bus. The packing and unpacking unit does not require the DMA channel to re-read the data for the next DMA transfer.

DMA Chaining Operation

The application programmer controls the DMA channels through chain descriptors. Each chain descriptor contains the following information:

- Next chain descriptor address pointer
- PCI address
- PCI upper address (for DAC DMA transactions)
- 80960 local bus address
- Byte count
- Descriptor control

The data contained in each chain descriptor provides the control information required for the DMA channel to perform the data transfer. Linking multiple chain descriptors allows the system to support, scatter and gather DMA. Creating multiple chain descriptors enables the application to gather non-contiguous blocks of memory. Once the DMA performs the transfer to/from the host memory, the data arrangement consists of a single block of contiguous data. Figure 18 shows an example of the DMA chaining operation.

Each descriptor contains a control value to notify the DMA channel of specific operations. For example, the control value specifies the PCI cycle type and when to generate an interrupt to the 80960RP processor. Control bits specify the interrupt mechanism which include:

- Continue to next chain descriptor without generating an interrupt
- Interrupt the 80960RP processor and continue with the next chain descriptor
- Interrupt on the end of the chain
- Interrupt on error conditions

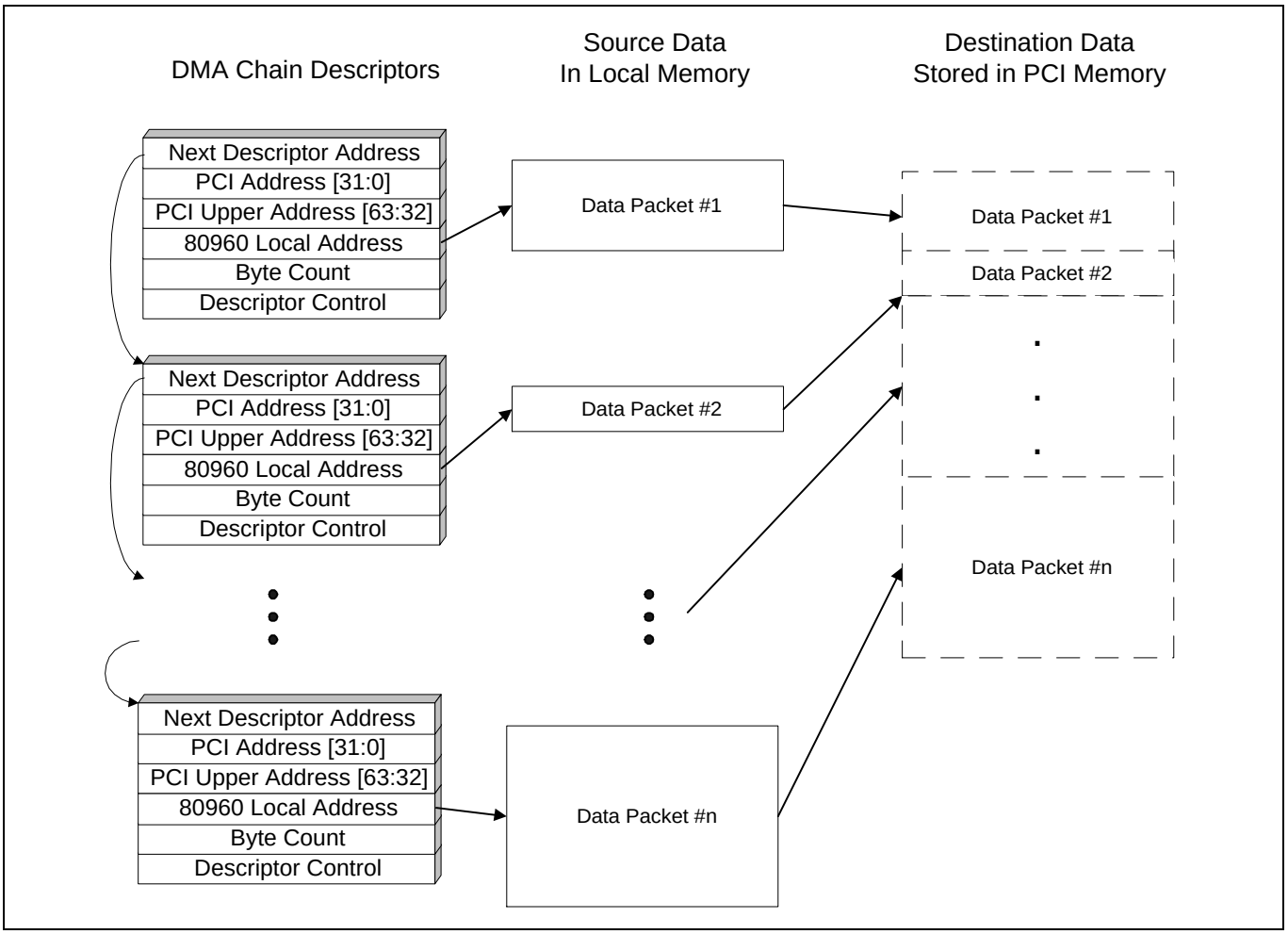

Figure 18. DMA Chaining Example

Because the DMA channels perform data transfers that are asynchronous to the processor operation, the application may need to append additional chain descriptors to the end of the chain list while the DMA channels perform data transfers. Each DMA channel supports appending chain descriptors in real-time, dynamic linking, without requiring the application to stop the DMA channel. Appending chains while the DMA channel is operating on the last descriptor or completed the entire chain list is permissible.

DMA Channel Queues

Each DMA channel of the i960 RP processor contains a 64-byte queue. These queues improve the system throughput by allowing the transactions to operate simultaneously on both the PCI bus and the 80960 local bus.

The DMA queues support high-performance bandwidth on both the PCI bus and the 80960 local bus. The DMA queue implementation provides a FIFO-style architecture that allows simultaneous data streaming to and from the queue. This data streaming allows each DMA channel to sustain data transfers up to 132 Mbytes/sec.

During DMA transactions, the queues will latch the data as it enters the DMA channel. Upon receiving the first data, the DMA asserts the request for the target bus in an attempt to acquire the target bus prior to filling the DMA queue. When the DMA channel acquires the target bus, the DMA will stream the data between the PCI bus and the local bus without breaking the transaction into multiple transactions.

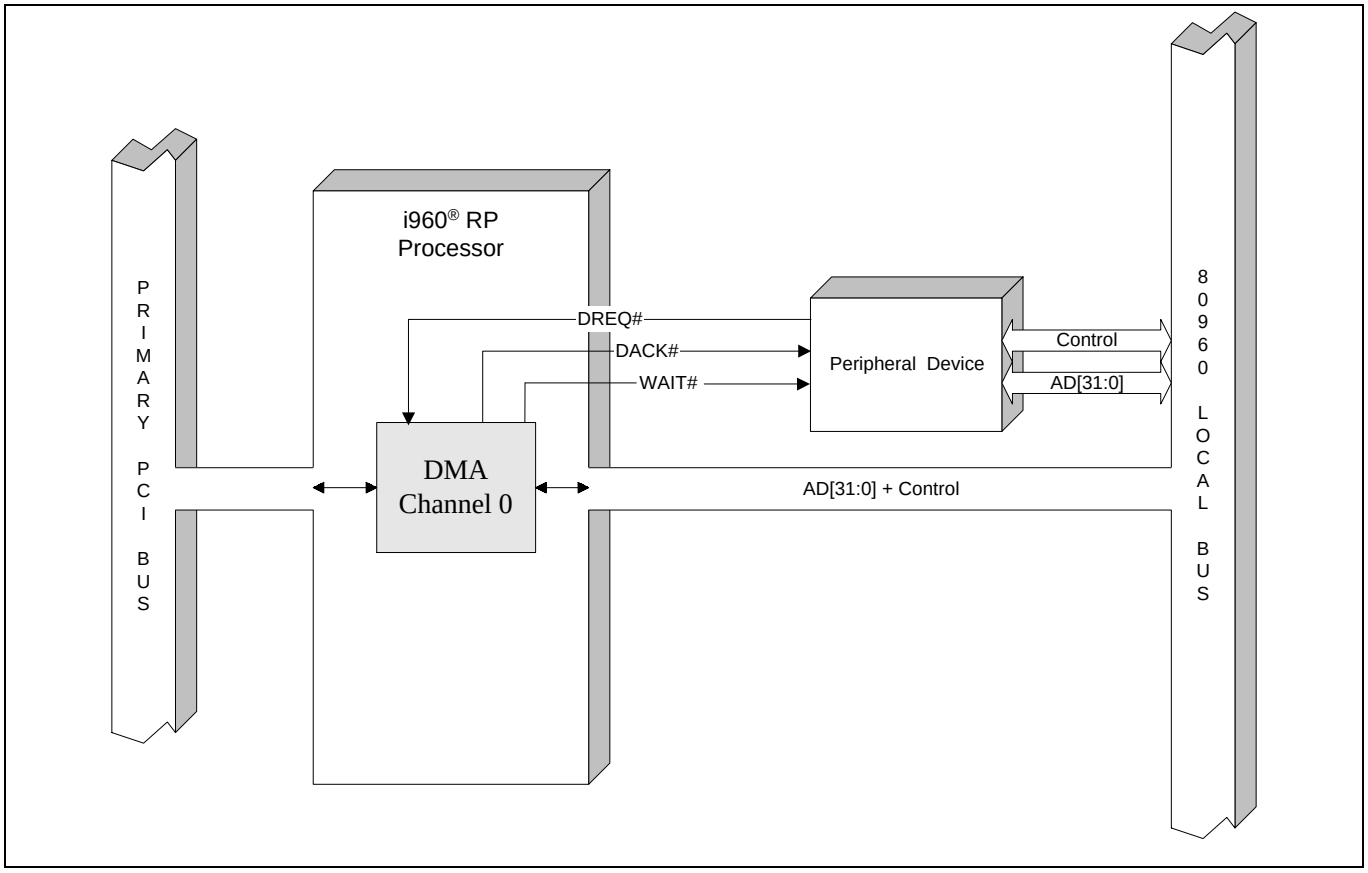

Figure 19. Demand Mode DMA Example

DMA Transactions

Each channel of the DMA controller has the flexibility to generate any of the PCI bus cycle types. This includes:

- *Memory read or memory read multiple* commands
- *Memory write* command
- *Memory write and invalidate* command

The DMA controller provides specialized support to ensure the *Memory Write and Invalidate* (MWI) command protocol. When the application programs the DMA channel to perform a MWI cycle, the DMA channel ensures the cycle meets the PCI protocol. This specialized hardware verifies the amount of data transferred during each transaction by reading (from the 80960 local bus) at least one cache line prior to initiating the PCI cycle. If for some reason the DMA channel cannot provide a full cache line, the channel automatically converts the cycle type to *a memory write* command.

The DMA controller also allows generation of dual address cycles on the PCI bus to access the full 64-bit address space.

Demand Mode DMA Transfers

DMA channel 0 provides a two-pin interface that supports DMA transfer to and from external devices on the 80960 local bus. This interface consists of a **DREQ#** pin that the external device asserts to signify new data to transfer or that it has available buffers for DMA transfers to the device. The second pin, **DACK#**, notifies the device that the DMA channel can receive additional data or it has data to send to the device.

Demand mode DMA transfers require the external device connected to the 80960 local bus to support the local bus control signals. Figure 19 shows an example of a demand mode DMA system.

BUS ARBITRATION SUPPORT

The i960® RP processor provides four internal arbitration blocks as shown in Figure 3. This arbitration logic controls the bus mastership for the two PCI buses and the 80960 local bus. The four arbitration units include:

1. Local bus arbiter
2. Secondary PCI bus arbiter for external secondary PCI bus masters
3. Primary PCI bus arbiter for internal primary PCI bus masters
4. Secondary PCI bus arbiter for internal secondary PCI bus masters

Local Bus Arbitration Unit

The 80960 local bus arbitration unit (LBAU) supports up to seven local bus masters. The bus masters control the local bus when granted by the local bus arbiter. The local bus masters include:

- 80960RP Processor
- DMA channel 0
- DMA channel 1
- DMA channel 2
- Primary ATU (for inbound address transactions)
- Secondary ATU (for inbound address transactions)
- External local bus master

The arbitration logic controls the bus mastership of the 80960 local bus. When a bus master requests the local bus, the local bus arbitration unit obtains control of the 80960 local bus from the 80960RP processor. Once the LBAU receives control, the LBAU grants the requesting master the local bus. If the processor requires ownership of the local bus after granting the bus to another master, the processor will notify the LBAU. The LBAU will place the 80960RP processor request in the arbitration algorithm.

The LBAU supports three priority levels of round-robin arbitration. The three levels define the low, medium, and high priorities. The round-robin mechanism ensures that for each priority level there will be a winner. Every arbitration cycle, a LBAU will either grant the bus or promote a masters request to the next higher priority level. The LBAU reserves one slot on level 00_2 and level 01_2 for the device promoted from the lower priority level. By reserving this slot, the algorithm still guarantees fairness by allowing lower priority requests to be promoted. The LBAU promotes the device until it becomes the highest priority device and is granted the bus next. The LBAU demotes the device to its original programmed priority after the device completes the transaction.

The LBAU contains a programmable 12-bit counter used to limit the length of time each bus master may own the 80960 local bus. The counter is programmable from the 80960RP processor. This counter will decrement on each 80960 processor clock. The LBAU automatically reloads the counter after granting ownership of the 80960 local bus. Upon reaching a zero count, the arbitration logic will notify loss of the local bus to the existing bus master. The bus master will gracefully complete the current transaction. If no pending bus request exists, the current bus master will retain ownership of the 80960 local bus and the counter will remain at a zero count. Figure 20 shows a local bus arbitration example.

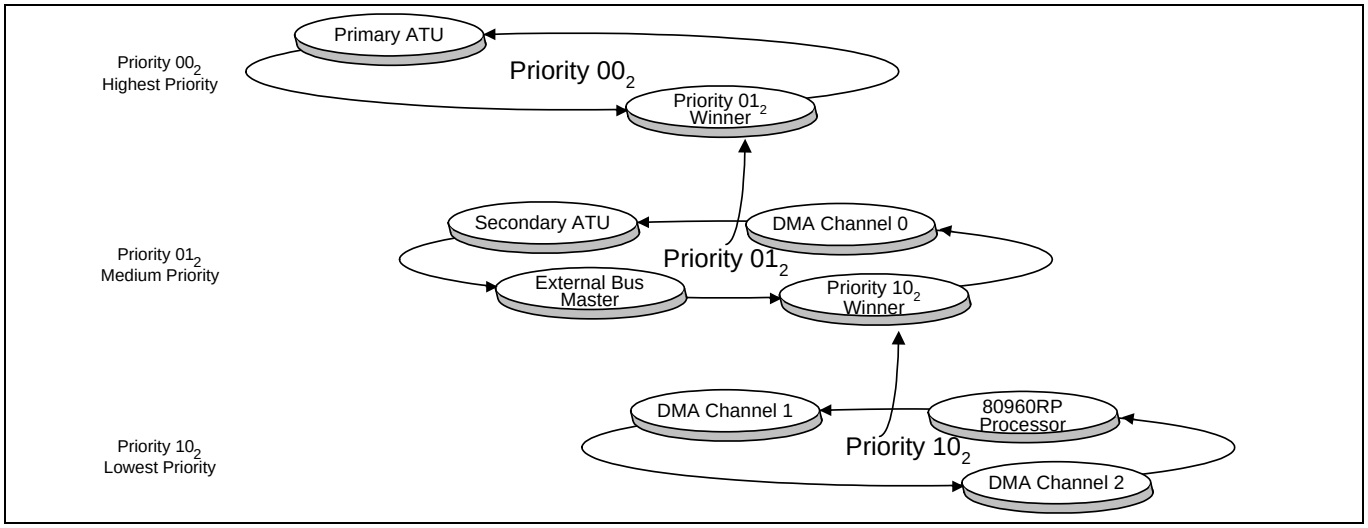

Figure 20. Local Bus Arbitration Example

Secondary PCI Bus Arbitration Unit

The secondary PCI bus arbitration unit (SAU) supports six external PCI bus masters on the secondary PCI bus and the secondary bus interface of the 80960RP processor. The PCI local bus specification defines the handshaking protocol when each bus master controls the secondary PCI bus when granted control by the SAU.

The arbitration logic supports granting control of the secondary PCI bus prior to the existing master completing its transaction. When a bus master requests the secondary PCI bus, the SAU will grant control to the requester who wins the arbitration cycle.

The SAU supports three priority levels of round-robin arbitration. The three levels define the low, medium, and high priorities. The round-robin mechanism ensures that for each priority level there will be a winner. In every arbitration cycle, an SAU will either grant the bus or promote the master to the next higher priority level request. The SAU reserves one slot on level 00_2 and level 01_2 for the device promoted from the lower priority level. By reserving this slot, the algorithm still guarantees fairness by allowing lower priority requests to be promoted. The SAU promotes the device until it becomes the highest priority device and is granted the bus when available. The SAU demotes the device to its originally programmed priority after the device completes the transaction.

The arbiter interfaces to all requesting agents on the bus through the **REQ#-GNT#** protocol. A bus master will assert it's **REQ#** output and wait for its **GNT#** input to be asserted. The SAU can grant an agent the bus while a previous bus owner still has control. The SAU decides only which PCI device to assign the bus to next. Each individual PCI device is responsible for determining when the bus actually becomes free and it is allowed to start its bus access.

Primary and Secondary Internal PCI Bus Arbiters

The 80960RP processor integrates two internal arbitration units which control access to the internal PCI buses within the device. The primary internal PCI arbitration unit arbitrates for the following internal units:

- Primary 80960RP bridge interface
- Primary Address Translation Unit
- DMA channel 0
- DMA channel 1

The secondary internal PCI arbitration unit arbitrates for the following internal units:

- Secondary 80960RP bridge interface
- Secondary Address Translation Unit
- DMA channel 2

Each internal PCI arbitration unit uses a fixed round-robin arbitration scheme with each device on a bus having equal priority.

Each PCI interface of the processor contains a master latency timer (MLT) for use by the internal resources when they are acting as PCI bus masters. Both ATUs, the DMA channels, and the bridge interfaces use an MLT. The MLT usage, as defined by the PCI specification, determines the minimum period of time the master is allowed to own the PCI bus. The internal PCI arbitration unit extends this concept by adding all of the internal bus resources to the arbitration equation and is capable of removing the current bus master when its time expires.

Each internal bus master may lose its grant for either of the two conditions:

1. If an external bus master wants the bus (external grant inactive), or
2. An internal bus master wants the bus (internal grant inactive while external grant still active).

The bridge configuration space and the ATU configuration space contain independent MLT registers. The bridge loads the MLT value into the internal arbitration counter only when granted ownership of the PCI bus. All other cases will load the MLT value from the ATU configuration space.

INTEGRATED MEMORY CONTROLLER

The i960® RP processor integrates a memory controller to provide a direct interface between the processor and a local memory subsystem. The memory controller (MC) supports connection of DRAM, SRAM, ROM and FLASH directly to the processor.

DRAM Control

The memory controller supports up to 256 Mbytes of DRAM. The controller supports from one to four banks of DRAM organized as 32-bits or 36-bits wide. A programmable option allows non-interleaved or two-way interleaved DRAM, depending on the memory type used. The MC supports three different architectures of DRAM: Fast Page-Mode (FPM), Extended Data Out (EDO), and Burst Extended Data Out (BEDO). The memory controller provides all of the control signals to support interleaved Fast Page-Mode DRAM. The memory controller provides a programmable DRAM refresh counter to generate the cycles necessary to refresh the charge in the DRAM cells.

The Memory Address Bus signals (**MA[11:0]**), the Row Address Strobes (**RAS[3:0]#**) and Column Address Strobes (**CAS[7:0]#**) control both read and write cycles. The memory controller presents the row address and column address over the **MA[11:0]** bus. The memory controller supports Fast Page-Mode early write cycles, which asserts **DWE[1:0]#** for the entire transaction.

The memory controller for DRAM provides a programmable address windowing mechanism that decodes local bus addresses and drives the corresponding DRAM control signals. The memory controller's memory-mapped registers control the address window. Additional memory-mapped registers control the timing for different speed ratings of DRAM, DRAM bank sizes, DRAM types, DRAM initialization and DRAM organization.

The DRAM organization is programmable through control bits in the DRAM bank control register. The MC provides support for up to four banks of non-interleaved DRAM. The MC supports up to two banks of interleaved DRAM with each bank containing one or two leaves. Figure 21 shows an example of non-interleaved DRAM.

The memory controller provides eight **CAS#** signals for the support of interleaved memory. The **CAS[3:0]#** signals provide the byte selection for one leaf, while the **CAS[7:4]#** signals provide it for the second leaf. It is necessary to control external buffer output enables during read transactions in an interleaved memory system. The MC provides two signals (**LEAF[1:0]#)** to control the multiplexing of data from each memory leaf onto the processor address/data bus. The system designer uses these signals for the **OE#** pins of the data transceivers in an interleaved memory array. The MC supports widely accepted DRAM device sizes from 1 Mbit to 64 Mbit without using external logic to generate control signals. The MC provides two identical write enable signals (**DWE[1:0]#**) to control the **WE#** input of DRAM devices during read and write transactions. An example of a memory system consisting of 32-bits of interleaved DRAM without parity is shown in Figure 22.

Programmable Refresh Timer

The DRAM memory cell retains data in its correct state by maintaining power and executing a **RAS#** refresh cycle. The memory controller supports **CAS#** before **RAS#** (CBR) refresh cycles, a commonly implemented refresh mechanism. CBR refresh requires no processor overhead and reduces logic requirements for DRAM refresh.

The MC uses the processor input clock to derive the time delay for CBR refresh cycles. The internal DRAM Refresh Interval Register provides the time delay between DRAM refresh cycles, and is programmed in increments of this clock. The register provides ten bits for the programmed value, which corresponds to a time delay range of:

- 0 to 40.92 µsec at 25 MHz
- 0 to 34.1 µsec at 33 MHz

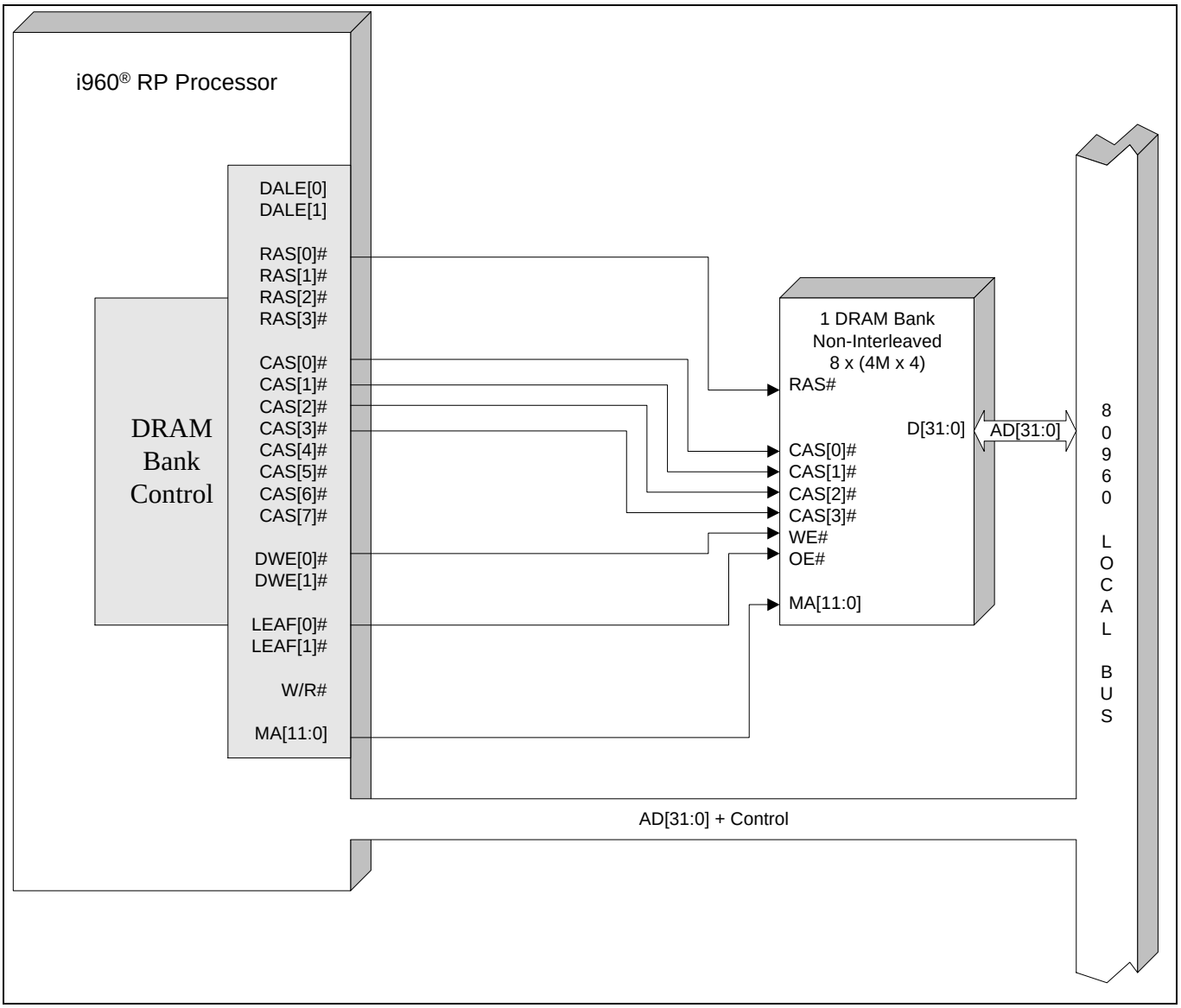

Figure 21. Non-Interleaved DRAM Example

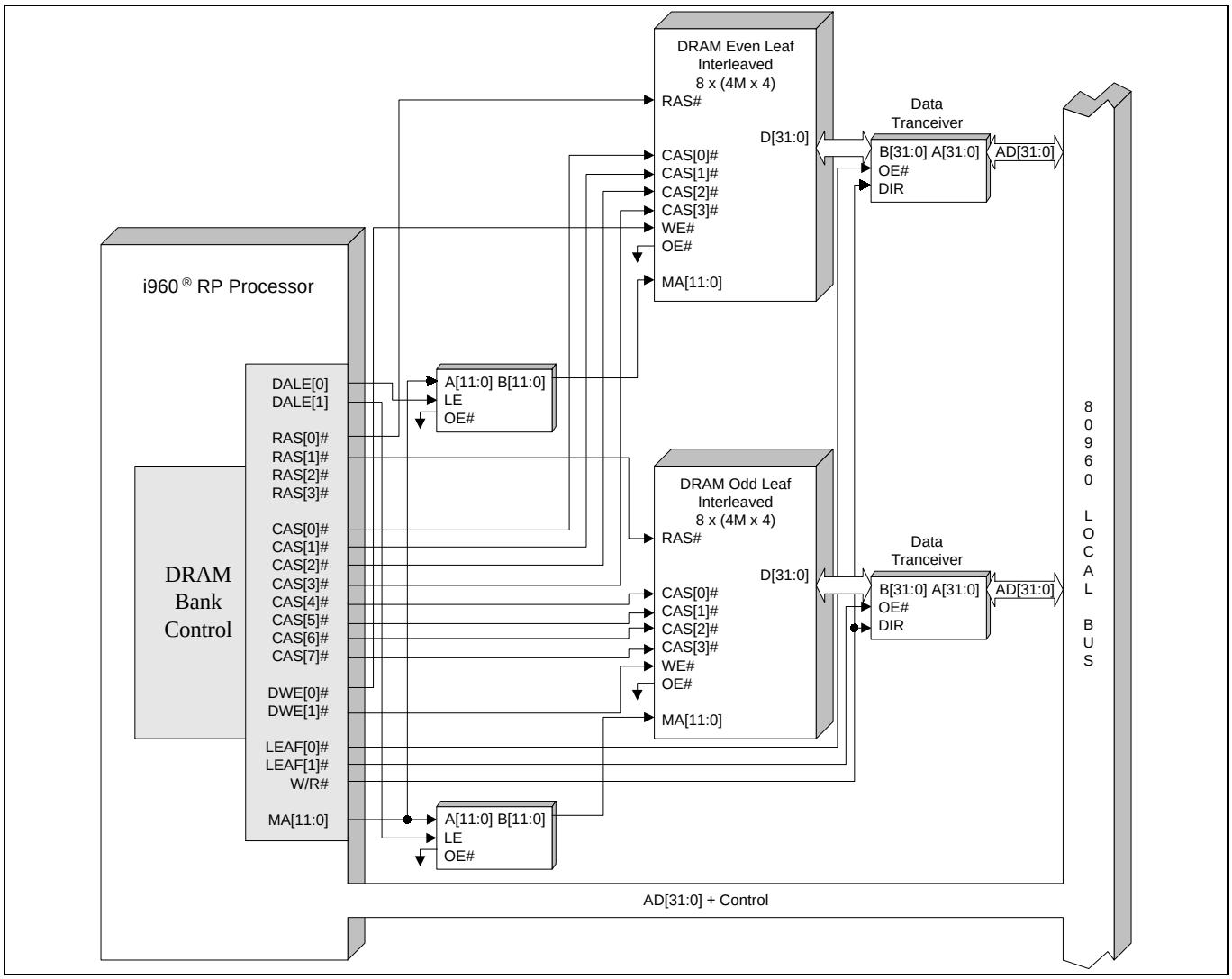

Figure 22. Interleaved DRAM Example

DRAM Performance

The optimized DRAM interface of the 80960RP memory controller achieves high performance from low-cost memory systems. The following shows the performance achievable by implementation of various DRAM technologies.

60 ns DRAM Interleaved & Buffered, 33 MHz

- Read: A 1 2 D D D D D...
- Write: A 1 D D D D D...

70 ns DRAM Non-Interleaved, 25 MHz or 33 MHz

- Read: A 1 2 D 1 D 1D 1 D 1 D...
- Write: A 1 2 D 1 D 1 D 1 D 1 D...

70 ns EDO DRAM Non-Interleaved, 25 MHz

- Read: A 1 2 D D D D D...
- Write: A 1 D D D D D...

50 ns BEDO DRAM Non-Interleaved, 33 MHz

- Read: A 1 2 D D D D D...
- Write: A 1 D D D D D...

Memory Controller Error Reporting

The MC provides two mechanisms for reporting error conditions. The first is DRAM parity and the second is a bus monitor used to detect invalid local bus addresses.

Programmable Byte Parity for DRAM

The selection of parity is programmable by the application through the internal DRAM Parity Enable Register. On reset, the MC disables parity. When the application enables data parity, the memory controller generates the selected parity for each byte written to DRAM and presents it to the parity bus (**DP[3:0]**). Parity is checked on all DRAM read accesses when enabled.

Upon detection of a parity error, the MC latches the 30-bit address of the faulty memory location and stores the address in the Memory Error Address Register. The MC will detect parity errors for any local bus master. These include the primary ATU, secondary ATU, DMA channel 0,1 or 2 and the 80960RP processor. The memory controller detects when the 80960RP processor is the bus master, sets the parity error status bit in the local processor status register, and generates an NMI# interrupt. When the processor is not the bus master, the memory controller will notify the other bus masters of the error condition. Those masters will latch the error and generate an NMI# interrupt to the 80960 processor.

When the 80960RP processor detects an interrupt, the interrupt service routine will read the NMI# latch to determine the source of the interrupt. The interrupt service routine can read the memory error address register to isolate the cause of the parity error.

Bus Monitor Support

The MC contains logic that monitors all bus accesses to any memory region configured for external memory control. If an external memory controller does not assert the **RDYRCV#** signal to terminate the access, the processor will stall. The MC always enables the bus monitor. However, the application can enable or disable the interrupt generated to the 80960RP processor from the memory controller. The bus monitor operates by monitoring a combination of the internal signals. The external memory controller has up to 127 CLKIN periods to assert **RDYRCV#**. Otherwise the memory controller assumes a bus fault condition and terminates the transfer. The MC asserts bus fault signal to the local bus masters or asserts to the local processor error signal from the memory controller.

SRAM, Flash and ROM Control

The MC supports two independent banks, Memory Bank 0 and Memory Bank 1, to connect ROM, SRAM or Flash memory devices to the processor. The MC supports devices organized as 8-bit or 32-bit wide memory. Each SRAM/ROM bank has a programmable window of local bus addresses that can be programmed to correspond to any local bus address. Memory banks must not overlap with reserved addresses. Each SRAM/ROM bank will assert its respective chip enable signal (**CE[1:0]#**) when the address on the local bus falls within the programmed window for the SRAM/ROM bank. The SRAM/ROM banks have independent control to support different memory types in each bank. The memory write enable signals (**MWE[3:0]#**) provide the write strobes for the selected memory bank. Connecting SRAM/ROM to the MC requires a combination of MC signals and local bus signals.

The MC uses the **MA[11:0]** pins to provide a 12-bit burst address when the memory access falls within the address windows for both Memory Banks 0 and 1. For 32-bit wide memory, the **MA[11:0]** pins will latch the address and provide an incrementing address during burst data accesses. The **MA[11:0]** will increment for burst data transfers of up to a 2 Kbyte boundary.

Eight bit wide memories have a maximum burst count of four accesses. The **BE[1:0]#** pins, which translate to **A[1:0]**, provide the incrementing burst address. Figure 23 shows an example of an 8-bit memory connected to the 80960RP processor.

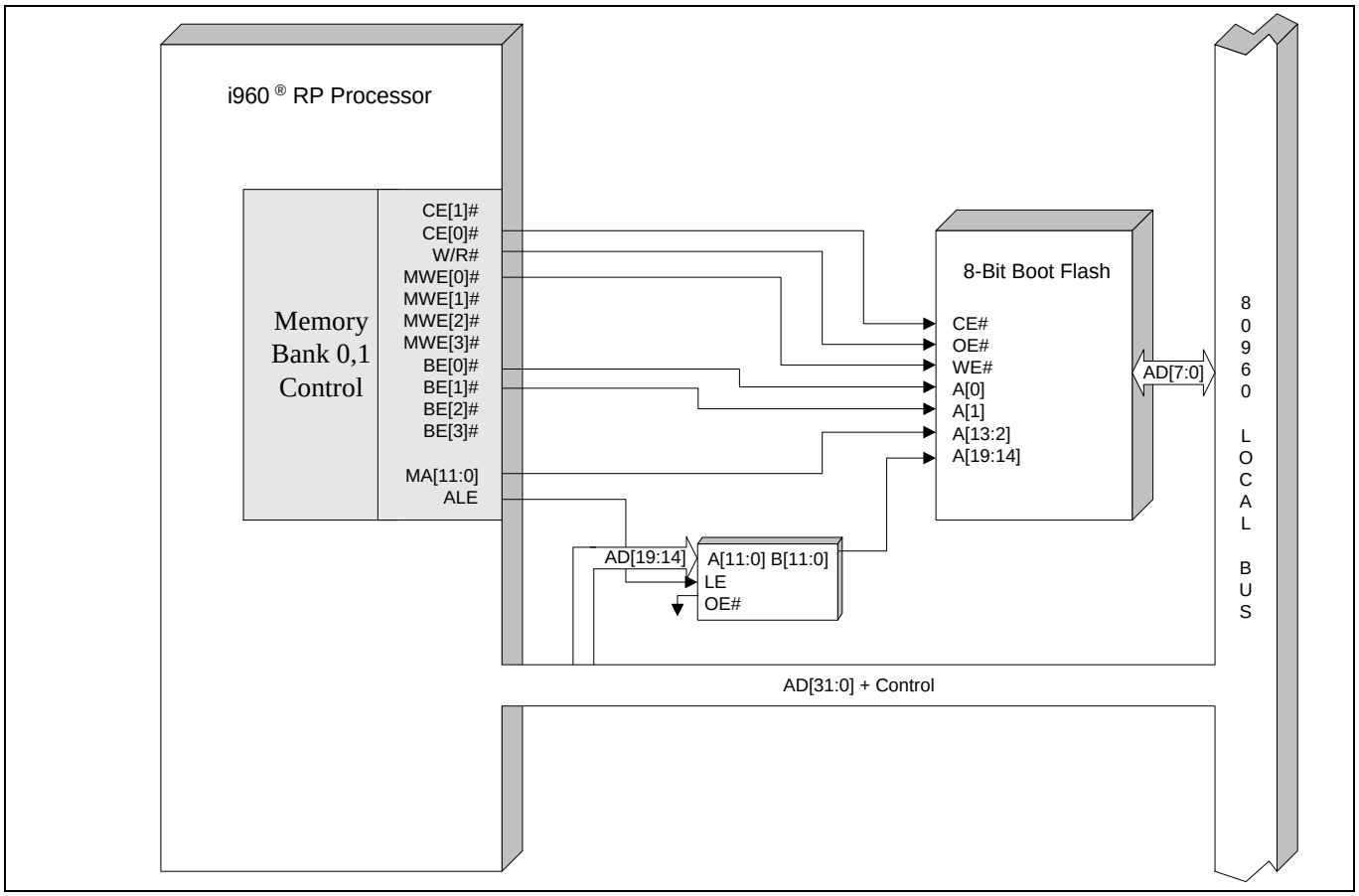

Figure 23. 8-bit Memory Example

During ROM, SRAM and flash memory accesses, the MC generates the burst address bits in conjunction with the control signals. The MC generates the lower twelve bits of the address on the **MA[11:0]** Memory Address Bus. The local bus master generates the upper address bits on the **AD[31:14]** multiplexed Address/Data bus. When addressing 8-bit memory **BE[1]#** becomes **A[1]** and **BE[0]#** becomes **A[0]** as shown in Figure 23. Since the memory controller only latches **A[13:2]**, external logic must use **ALE** to latch the upper address bit during an address cycle. The MC provides unique chip enable signals (**CE[1:0]#**) to select the device and activate its control logic during a memory access.

The MC provides the memory write enable signals (**MWE[3:0]#**) to select the individual byte lanes during memory write accesses. During a memory write access the MC asserts a combination of **MWE[3:0]#** and **CE[1:0]#** for the data cycle. The bus master drives the W/R# signal HIGH to prevent the device's output from being enabled onto the address/data bus.

The **MWE[3:0]#** signals control the selection of individual byte-wide flash memory devices during programming without the use of external logic. The memory write enable bit allows the memory controller to assert the **MWE[3:0]#** during write cycles. The application software accesses this bit through the Memory Bank Control Register. If the application uses memory banks 0 or 1 for SRAM, the memory write enable bit must be set to enable the assertion of the **MWE[3:0]#** signals for memory write transactions. Refer to Figure 24 for an example of a 32-bit flash memory system connected to the 80960RP processor.

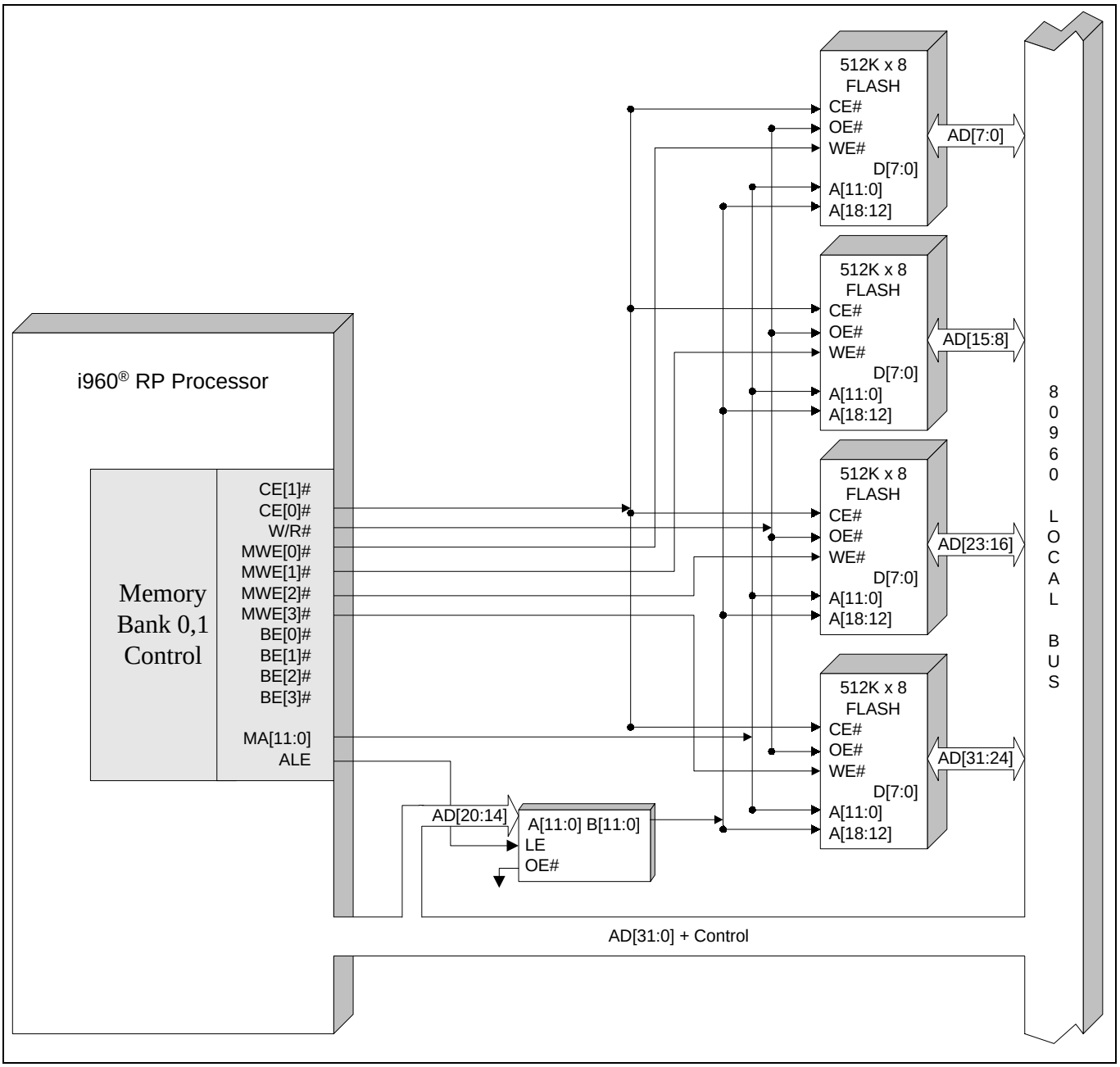

Figure 24. 32-bit SRAM/Flash Memory Example

FILTERING PCI INTERRUPTS

The i960® RP processor allows improved system performance by providing the intelligence to handle the PCI device interrupts at the subsystem level. With the i960 RP processor, the I/O interrupt provides:

- Reduced upstream interrupts
- Freedom from low-level interrupt processing by the host CPU
- Reduced interrupt latency
- Efficient PCI bandwidth usage
- Creation of an intelligent upstream I/O subsystem

The i960 RP processor design integrates hardware to support the PCI interrupt structure. This hardware enables the 80960RP processor to filter the PCI interrupts and selectively perform the interrupt processing, or to pass the interrupt upstream to the host processor.

The interrupt hardware works in conjunction with the 80960RP processor's integrated interrupt controller. Figure 25 depicts the supported functions.

The PCI interrupt steering support in the i960 RP processor allows the system designer to route the interrupts directly to the PCI **INTA#**, **INTB#**, **INTC#**, and **INTD#** pins without the use of any external configuration jumpers. The software can then direct all PCI interrupts upstream to the host processor or to the 80960RP processor.

The 80960RP processor can generate any PCI interrupt **INTA-D#**. This allows the 80960 processor to determine exactly which PCI device on the **INTA-D#** pins generated the interrupt, process the interrupt, or forward the interrupt to the host processor.

The 80960RP processor supports generating interrupts upstream to the host processor via two different methods. First, the 80960 processor contains a hardware port that allows the 80960 software to generate an interrupt directly on any **INTA-D#** pins. The second method allows the 80960RP processor to generate interrupts directly to the host processor over the APIC bus.

I/O APIC Interface

The 80960RP processor provides a 3-wire I/O Advance Programmable Interrupt Controller (APIC) port. The I/O APIC interface unit utilizes the interrupt input pins on which I/O devices inject interrupts into the system as an edge or level. Through APIC emulation software using the I/O APIC port, the 80960RP processor provide interrupt functionality via the APIC bus. The I/O APIC emulation software supports a re-direction table with an entry for each interrupt input pin. Each entry in the re-direction table can be individually programmed to indicate whether an interrupt on the pin is recognized as level sensitive; the vector and priority for the interrupt; and which of all possible processors should service the interrupt and how to select that processor. The emulation software uses the information in the table to broadcast a message to all local APIC units.

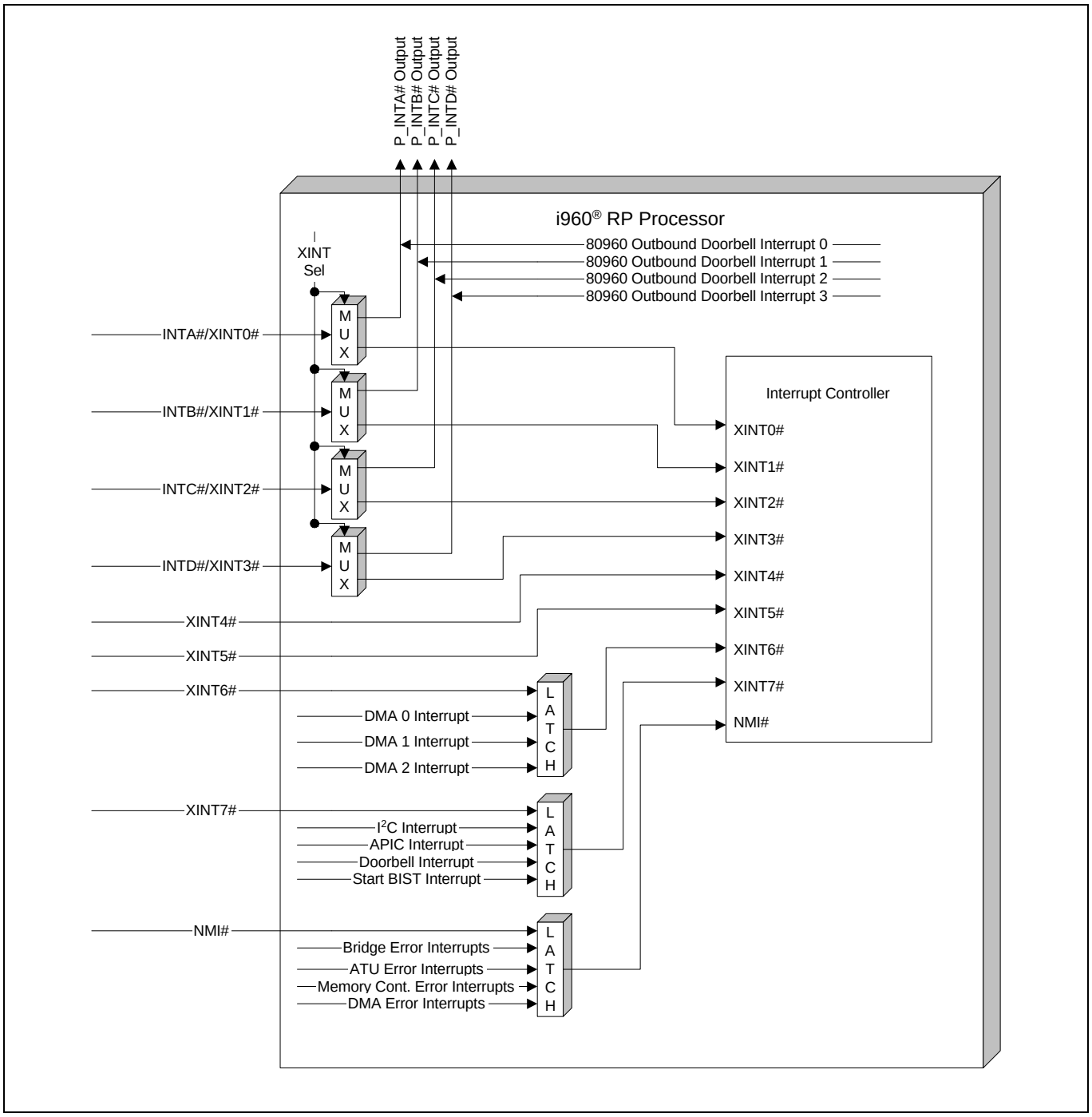

Figure 25. PCI Interrupt Steering

SERIAL I²C INTERFACE

The i960® RP processor integrates a serial I²C bus interface unit. The I²C bus interface unit allows the 80960RP processor to serve as both a master and slave device residing on the I²C bus. The I²C bus, developed by Philips Corporation, consists of a two-pin interface and capable to support frequencies up to 400 KHz.

The I²C bus allows the i960 RP processor to interface to other I²C peripherals and microcontrollers for system management functions. This serial bus defines a complete serial protocol for passing information between agents on the bus using only a two-pin interface. A unique 7-bit address identifies each device on the bus and operates as a master (transmitter) or as a slave (receiver).

The I²C bus allows for a multi-master system, which supports multiple devices simultaneously. The I²C bus defines an arbitration procedure to handle this scenario.

Some of the server management functions performed via an I²C bus include:

- Remote management
- Temperature monitoring
- Power supply monitoring
- Fan monitoring
- LCD display control

The 80960RP processor supports both speed definitions for the I²C bus, including 100 KHz and 400 KHz operations. The 80960RP processor provides the I²C clocks by dividing the processor clock cycles with a programmable clock generator.

The I²C unit consists of a three register-interface in addition to an interrupt mechanism which generates an interrupt to the 80960RP processor on transmit buffer empty and receive buffer full conditions.

i960® RP PROCESSOR CLOCKING

The i960® RP processor supports PCI operations and 80960 core processor frequencies up to 33 MHz. The on-chip peripheral units support synchronous clocking across two independent clocking boundaries. The 80960RP processor has two clock input pins that support:

- Primary PCI bus
- Secondary PCI bus, 80960 CPU, and local memory

This synchronous mode of operation bypasses any clocking boundaries. All units of the 80960RP processor operate at the same frequency to reduce data transfer latency. In addition, controlling the 80960 CPU and local memory with the secondary PCI bus clock reduces the number of clock sources required by the system to operate the 80960RP processor and memory subsystem.

PCI Configuration

The i960 RP processor is a multifunction PCI device with two PCI functions. Each function requires its own PCI configuration space. The host processor performs the PCI configuration from the primary PCI bus. The host processor is also responsible for initializing various registers within the i960 RP processor for PCI operations. However, the 80960RP processor can initialize the configuration registers before the host processor accesses the 80960RP processors configuration registers to determine the system requirements. All registers within the 80960RP processor's configuration space have read/write privileges from the i960 RP processor, even if they are read only from the PCI bus.

Access to the PCI configuration spaces requires a combination of hardware decode logic and a configuration cycle present on the primary PCI bus. The PCI specification defines a primary PCI interface pin (called **IDSEL#**) used to identify the device during a PCI configuration cycle. Figure 26 shows an example connecting the **IDSEL#** to **P_AD[18]**. The system design connects the **IDSEL#** signal to any one of the **P_AD[31:11]** address signals. The PCI local bus specification provides a complete description of configuration cycles generated on the PCI bus.

The host processor selects each individual PCI device within the 80960RP processor by the configuration cycle generated on the PCI bus to the i960 RP processor. The function numbers separate the configuration spaces within the i960 RP processor. Function #0 controls the configuration space of the PCI-to-PCI bridge. Function #1 controls the configuration space for accessing the 80960RP memory, the bus mastership portion of the Address Translation Unit, as well as the processors DMA controllers.

Peripheral Programming Interface

All of the i960 RP processor's integrated peripherals contain a memory-mapped interface for programming. This memory-mapped interface utilizes a small portion of the 80960 processor's local bus address space. The peripheral units include the following:

- DMA controller
- Memory controller
- 80960 local bus arbitration
- Address translation units
- Doorbell address window
- I^2C unit
- I/O APIC interface unit

In addition to the peripherals, the 80960RP processor provides accessibility to the PCI configuration registers. The memory-mapped interface provides the ability for the 80960RP processor to preset specific registers prior to the host processor's performing PCI configuration cycles to access the configuration space. This mechanism allows the 80960RP processor to set the memory limit registers in order to specify the amount of PCI address space, the device identification registers, and the expansion ROM registers to mention a few.

Reset Configuration Options

To ensure the 80960 processor completes the PCI register initialization prior to the host processor performing configuration cycles the 80960RP processor initialization options support two modes of processor boot modes.

The first mode holds the 80960 processor in reset until the host processor releases it. This mode allows the host processor (after a hardware reset on the PCI reset pin) to download any information to the 80960RP processor memory, configuration space, or peripheral memory-mapped registers. All units within the 80960RP processor operate in a fully functional mode. The reset modes only restrain the 80960RP processor core incorporated within the 80960RP processor.

The second mode allows the i960 RP processor to boot, including the 80960RP processor core. This requires the memory subsystem to contain some form of boot ROM for the 80960 processor to perform its initialization sequence. This special mode restricts the host processor from performing any configuration cycles or bridging operations by initiating a PCI **RETRY** before the 80960RP processor allows the primary PCI interface to accept PCI cycles.

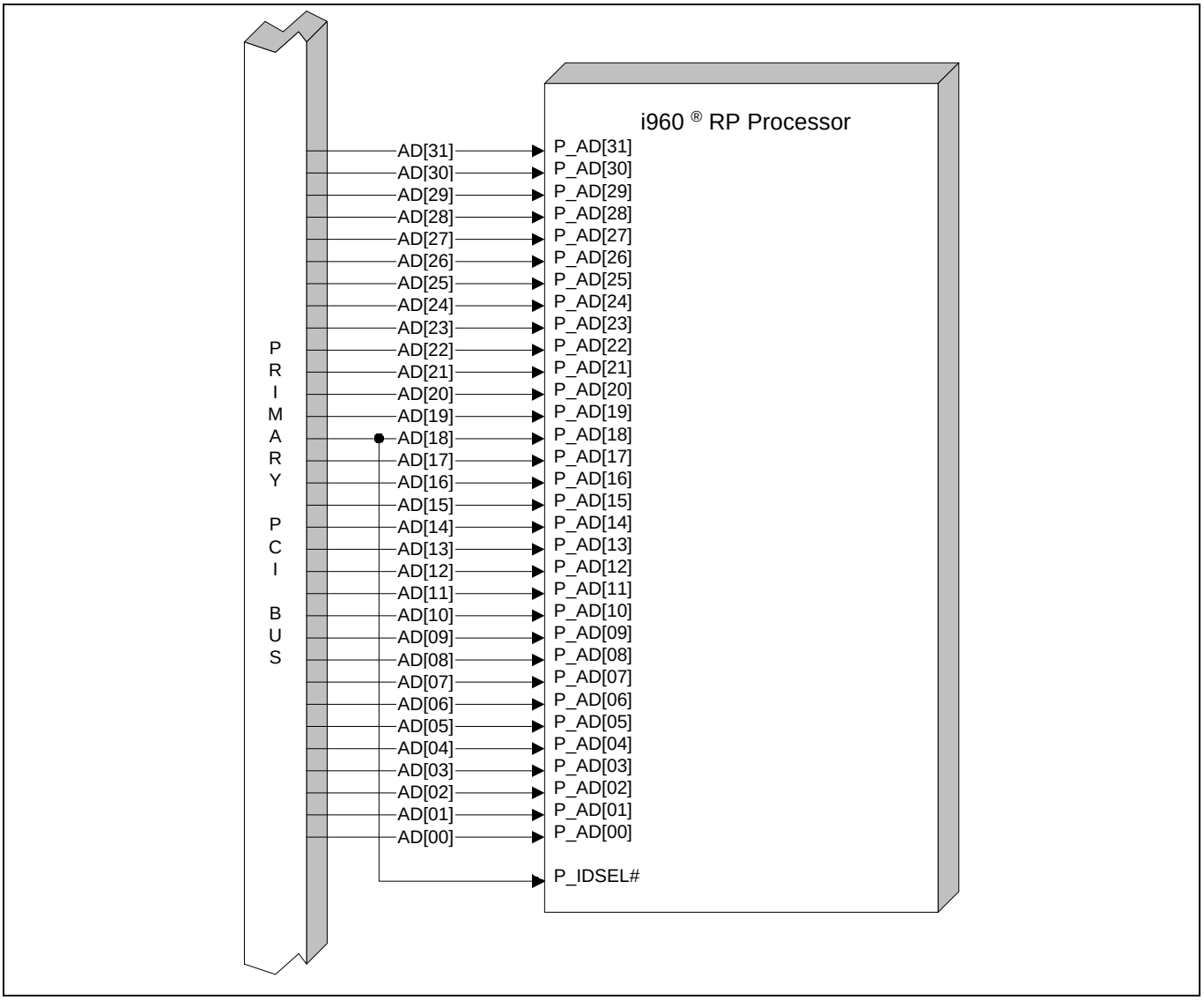

Figure 26. Connecting the IDSEL# Signal Example

i960® RP PROCESSOR PACKAGING

The i960® RP processor requires a high pin-count package. However, the package retains the low cost attributes of Plastic Quad Flat Pack (PQFP) packaging. For this reason, the Perimeter Ball Grid Array (BGA) package meets these requirements. The 352-BGA package provides the high pin count, power dissipation, and low cost for the intelligent I/O applications.

The BGA package technology uses solder balls distributed on the bottom of the package. This distribution provides higher lead counts in a space smaller than a QFP. The surface tension characteristics of the solder balls cause them to self-align during the manufacturing process. As the solder melts, the two surfaces—the balls and pads—pull together. This means that component alignment is not as critical as with PQFPs. Studies show BGA packages placed within $\pm$ 0.3048 mm align with the target board, whereas a similar PQFP must be within $\pm$ 0.0762 mm.

The BGA package, with a pitch of 1.27 mm between balls, simplifies manufacturing issues compared to the 0.65-mm fine pitch PQFP packages. In addition, the defect rate due to bent leads becomes non-existent. The high ball count (352 balls) does not require large amounts of board space. The 352-BGA package is approximately 35 x 35 mm, which is slightly smaller than a 196-PQFP (38 x 38 mm) package.

The i960 RP processor supports 5.0V PCI systems and dissipates less than 3 Watts at 33 MHz.

COMPLETE TOOL SET FOR EMBEDDED DESIGN

The i960® RP processor operation provides a complete solution to meet the demands of today's customers. There are more than 100 different products and tools available today from the Solutions960® program vendors to support the i960 architecture. Because the i960 RP processor incorporates the i960 JF processor core, most existing tools are appropriate.

State-of-the-Art Compiler Technology

The two-pass, profile-driven compilers use profile program information collected during the dynamic execution of code to drive the optimization strategy for the compiler. The goal is to recognize the run-time behavior of the program and to optimize the code for that behavior. To gather run-time statistics, the compiler first instruments the program. The programmer runs this program on the target with any number of representative data sets. Collection of the profile information about the program occurs during program execution. The programmer feeds the execution information back to the host development system used to optimize the code during a second pass of the compiler. Two-pass, profile-driven compilers enable many performance enhancing optimizations. These optimizations include inter-module inlining; rearrangement of basic blocks; setting branch prediction bits; the allocation of global variables to on-chip data RAM; the refinement of alias information; superblock formation; and many superblock-based optimizations.

Intel provides two tool chains for the i960 product line. The gcc960 compiler, based on the Free Software Foundation's GNU compiler, and the iC960 compiler based on Intel compiler technology. The current versions of both the gcc960 compiler and the iC960 compiler incorporate two-pass, profile-driven optimizations.

Integrated Software Debuggers

Software debuggers give the applications programmer the ability to probe into a system during program execution. The i960 architecture took this concept further by integrating trace capabilities on-chip, providing the debugger with tracing support. Tracing the software flow without the aid of an external tool such as a logic analyzer or emulator benefits the software debug cycle. Integration of the four breakpoint registers gives the software debugger the ability to provide hardware breakpoint capabilities with a software package.

Intel provides a software debugger for its i960 processors. The db960 software debugger provides the applications programmer with tracing and breakpoint functionality. Additionally, this software package comes in a retargetable form, thus enabling the applications programmer to integrate this debugger into his application.

Wide Variety of Operating Systems

Operating system (or real-time system executives) needs vary from application to application. They vary from high-performance, multi-tasking environments to simple in-line code with external interrupt events. The i960 architecture provides the system with the capability to perform either of these application types. The user/supervisor model, with accessibility to the various stacks, gives operating systems the mechanisms for the multi-tasking systems.

Emulators and Logic Analyzers

Emulators provide the support necessary for debugging both the hardware and software applications. These support tools provide the hardware engineer with the capability to exercise the system parameters: address bus, data bus and control signals. Typically, the initial hardware/software integration requires this support. The i960 architecture has many emulator support tools for all of the i960 processors.

Logic analyzers provide increased value by allowing visibility into complex breaking conditions. In addition, the deep trace memory of the logic analyzer aids the applications engineer in viewing the history of the execution flow. An important factor of the logic analyzer lies in the fact that it is non-intrusive. The system runs at full speed with no performance degradation.

Evaluation Platforms

The i960 processor family contains many evaluation platforms. They range from multi-purpose platforms with various memory architectures to specific applications platforms designed for a single memory types. Each member of the i960 family provides an evaluation platform for the early software development prior to system hardware.

i960® Microprocessor PCI I/O Software Development Kit

Intel i960 microprocessor PCI I/O Software Development Kit (PCI-SDK) is a low-cost, scaleable development kit. The PCI-SDK can improve development time for emerging intelligent PCI I/O applications like ATM, Caching SCSI, RAID, and Fibre Channel.

The PCI-SDK features a software development platform with PCI Technology's PCI 9060 (a PCI to i960 processor bus bridge chip), an interchangeable Intel i960 processor module of your choice and an optional interchangeable I/O interface target module. The kit also includes a complete set of software development tools and related documentation to enable improved development time.

PCI Compliance

The i960 RP processor contains two PCI interfaces. These interfaces are compliant with the PCI Special Interest Group's specifications:

- *PCI Local Bus Specification* Revision 2.1

SUMMARY

The i960® RP processor exploits the advantages of PCI architectures by providing a high-performance processor with PCI specific integration. The i960 RP processor architectural requirements, taken from applications ranging from networking and storage add-in cards to server motherboard systems, comprise today's intelligent PCI I/O subsystem. The level of integration found in the i960 RP processor is achievable using Intel's highly advanced design and manufacturing technology, coupled with packaging technology exceeds other processors available for these PCI functions today.