High Performance I/O Design in the AlphaServer 4100 Symmetric Multiprocessing System

Samuel H. Duncan
Craig D. Keefer
Thomas A. McLaughlin

The DIGITAL AlphaServer 4100 symmetric multiprocessing system is based on the Alpha 64-bit RISC microprocessor and is designed for fast CPU performance, low memory latency, and high memory and I/O bandwidth. The server's I/O subsystem contributes to the achievement of these goals by implementing several innovative design techniques, primarily in the system bus-to-PCI bus bridge. A partial cache line write technique for small transactions reduces traffic on the system bus and improves memory latency. A design for deadlock-free peer-to-peer transactions across multiple 64-bit PCI bus bridges reduces system bus, PCI bus, and CPU utilization by as much as 70 percent when measured in DIGITAL AlphaServer 4100 MEMORY CHANNEL clusters. Prefetch logic and buffering supports very large bursts of data without stalls, yielding a system that can amortize overhead and deliver performance limited only by the PCI devices used in the system.

The AlphaServer 4100 is a symmetric multiprocessing system based on the Alpha 21164 64-bit RISC microprocessor. This midrange system supports one to four CPUs, one to four 64-bit-wide peer bridges to the peripheral component interconnect (PCI), and one to four logical memory slots. The goals for the AlphaServer 4100 system were fast CPU performance, low memory latency, and high memory and I/O bandwidth. One measure of success in achieving these goals is the AIM benchmark multiprocessor performance results. The AlphaServer 4100 system was audited at 3,337 peak jobs per minute, with a sustained number of 3,018 user loads, and won the AIM Hot Iron price/performance award in October 1996.[1]

The subject of this paper is the contribution of the I/O subsystem to these high-performance goals. In an in-house test, I/O performance of an AlphaServer 4100 system based on a 300-megahertz (MHz) processor shows a 10 to 19 percent improvement in I/O when compared with a previous-generation midrange Alpha system based on a 350-MHz processor. Reduction in CPU utilization is particularly beneficial for applications that use small transfers, e.g., transaction processing.

I/O Subsystem Goals

The goal for the AlphaServer 4100 I/O subsystem was to increase overall system performance by

- Reducing CPU and system bus utilization for all applications
- Delivering full I/O bandwidth, specifically, a bandwidth limited only by the PCI standard protocol, which is 266 megabytes per second (MB/s) on 64-bit option cards and 133 MB/s on 32-bit option cards
- Minimizing latency for all direct memory access (DMA) and programmed I/O (PIO) transactions

Our discussion focuses on several innovative techniques used in the design of the I/O subsystem 64-bit-wide peer host bus bridges that dramatically reduce CPU and bus utilization and deliver full PCI bandwidth:

- A partial cache line write technique for coherent DMA writes. This technique permits an I/O device to insert data that is smaller than a cache line, or block, into the cache-coherent domain without first obtaining ownership of the cache block and performing a read-modify-write operation. Partial cache line writes reduce traffic on the system bus and improve latency, particularly for messages passed in a MEMORY CHANNEL cluster.[2]
- Support for device-initiated transactions that target other devices (peers) across multiple (peer) PCI buses. Peer-to-peer transactions reduce system bus utilization, PCI bus utilization, and CPU utilization by as much as 70 percent when measured in MEMORY CHANNEL clusters. In testing, we ran a MEMORY CHANNEL application without peer-to-peer DMA, and observed 85 percent CPU utilization; running the same application with peer-to-peer DMA enabled, we observed 15 percent CPU utilization. The peer-to-peer technique is successfully implemented on the AlphaServer 4100 system without causing deadlocks.
- Large bursts of PCI-device-initiated DMA data to or from system memory. I/O subsystem support for large bursts of DMA data enables efficient PCI bus utilization because fixed bus latency can be amortized over these large transactions.
- Prefetched read data and posted write data buffering designed to keep up with the highest performance PCI devices. When used in combination with the PCI delayed-read protocol, the buffering and prefetching approach allows the system to avoid PCI bus stalls introduced by the bridge during PCI-device-initiated transactions.

The following overview of the system concentrates on the areas in which these techniques are used to enhance performance, that is, efficiency in the system bus and in the PCI bus bridge. In subsequent sections, we describe in greater detail the performance issues, other possible approaches to resolving the issues, and the techniques we developed. We conclude the paper with performance results.

AlphaServer 4100 System Overview

The AlphaServer 4100 system shown in Figure 1 includes four CPUs connected to the system bus, which comprises the data and error correction code (ECC) and the command and address lines. Also connected to the system bus are main memory and a single module with two independent peer PCI bus bridges. The single module, the PCI bridge module, provides the physical and the logical bridge between the system bus and the PCI buses. Each independent peer PCI bus bridge is constructed of a set of three application-specific integrated circuit (ASIC) chips, one control chip, and two sliced data path chips.

The two independent PCI bus bridges are the interfaces between the system bus and their respective PCI buses. A PCI bus is 64 or 32 bits wide, transferring data at a peak of 266 MB/s or 133 MB/s, respectively. In the AlphaServer 4100 system, the PCI buses are 64 bits wide.

The PCI buses connect to a PCI backplane module with a number of expansion slots and a bridge to the Extended Industry Standard Architecture (EISA) bus. In Figure 1, each PCI bus is shown to support up to four devices in option slots.

The AlphaServer 4000 series also supports a configuration in which two of the CPU cards are replaced with two additional independent peer PCI bus bridges. In the quad PCI bus configuration, there are 16 option slots available for PCI devices, at the cost of bounding the system to a maximum of two CPUs and two logical memory slots. This quad PCI bus configuration is shown in Figure 2.

Most of the techniques described in this paper are implemented in the PCI bus bridge. The partial cache line write technique, presented next, is also designed into the protocol on the system bus and into the CPU cards.

Improvements in CPU and System Bus Utilization through Use of Partial Cache Line Writes

Inefficient use of system resources can limit performance on heavily loaded systems. System designers must be attentive to potential performance bottlenecks beyond the commonly addressed CPU speed, cache loop time, and CPU memory latency. Our focus in the I/O subsystem design was to balance system performance in the face of a wide range of I/O device behaviors. We therefore implemented techniques that minimize the load on the PCI bus, the system bus, and the CPUs. The technique described in this section—partial cache line writes—reduces the load on the system bus and improves overall system performance.

Many first- and second-generation PCI controller devices were designed to operate in platforms that support 32-byte cache lines and 16-byte write buffers. It is common for an older PCI device to limit the amount of DMA data it reads or writes to match this characteristic of computers that were on the market at the time those devices were designed. Some classes of devices will, by their nature, always limit the amount of data in a burst transaction.

As do most Alpha platforms, the AlphaServer 4100 system supports a 64-byte cache line that is twice that of other common systems. When a PCI device performs a memory write of less than a complete cache line, the system must merge the data into a cache line while maintaining a consistent (coherent) view of

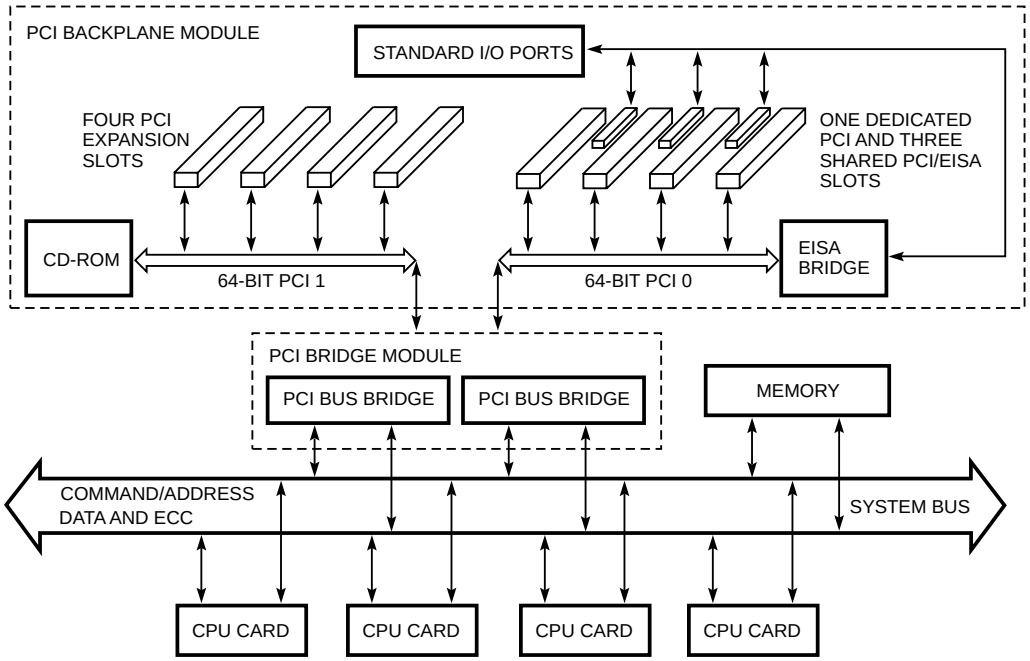


Figure 1
AlphaServer 4100 System with Four CPUs, Two 64-bit Buses

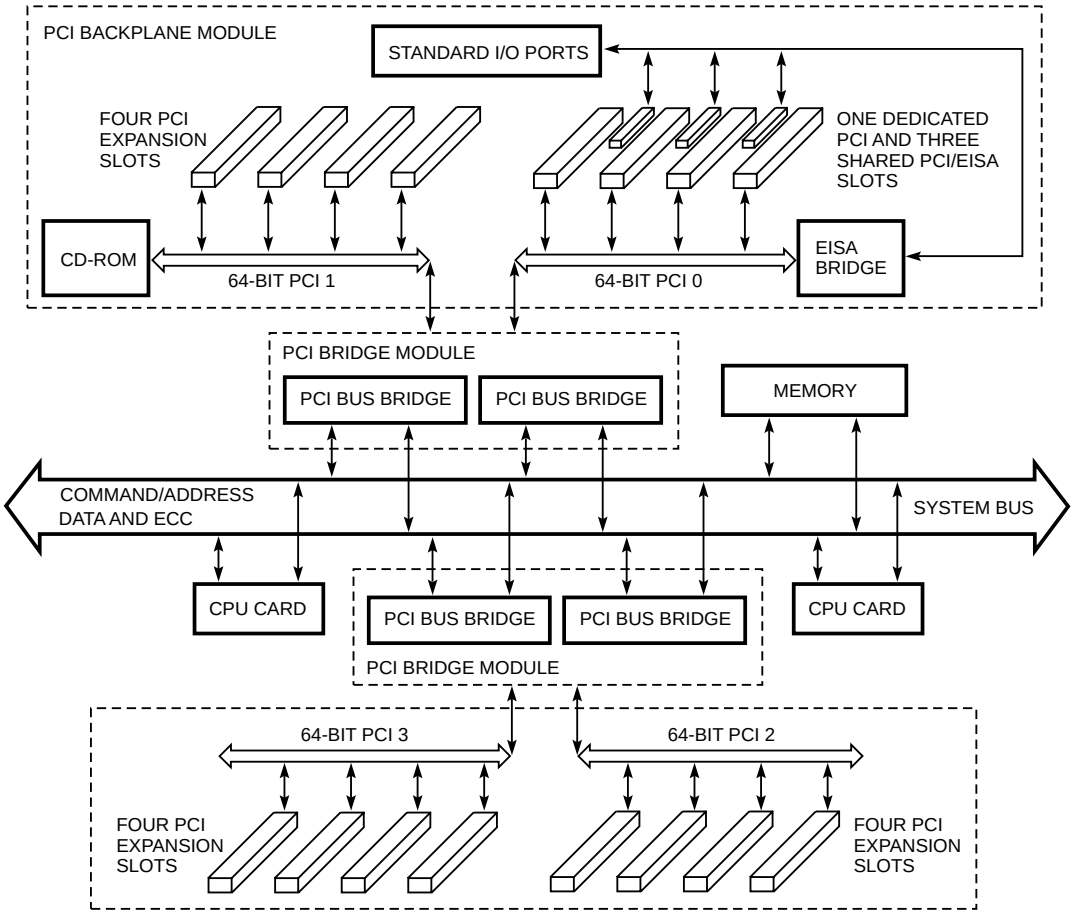


Figure 2
AlphaServer 4000 System with Two CPUs, Four 64-bit Buses

memory for all CPUs on the system bus. This merging of write data into the cache-coherent domain is typically done on the PCI bus bridge, which reads the cache line, merges the new bytes, and writes the cache line back out to memory. The read-modify-write must be performed as an atomic operation to maintain memory consistency. For the duration of the atomic read-modify-write operation, the system bus is busy. Consequently, a write of less than a cache line results in a read-modify-write that takes at least three times as many cycles on the system bus as a simple 64-byte-aligned cache line write.

For example, if we had used an earlier DIGITAL implementation of a system bus protocol on the AlphaServer 4100 system, an I/O device operation on the PCI that performed a single 16-byte-aligned memory write would have consumed system bus bandwidth that could have moved 256 bytes of data, or 16 times the amount of data. We therefore had to find a more efficient approach to writing subblocks into the cache-coherent domain.

We first examined opportunities for efficiency gains in the memory system.[3] The AlphaServer 4100 memory system interface is 16 bytes wide; a 64-byte cache line read or write takes four cycles on the system bus. The memory modules themselves can be designed to mask one or more of the writes and allow aligned blocks that are multiples of 16 bytes to be written to memory in a single system bus transaction. The problem with permitting a less than complete cache line write, i.e., less than 64 bytes, is that the write goes to main memory, but the only up-to-date/complete copy of a cache line may be in a CPU card's cache.

To permit the more efficient partial cache line write operations, we modified the system bus cache-coherency protocol. When a PCI bus bridge issues a partial cache line write on the system bus, each CPU card performs a cache lookup to see if the target of the write is dirty. In the event that the target cache block is dirty, the CPU signals the PCI bus bridge before the end of the partial write. On dirty partial cache line write transactions, the bridge simply performs a second transaction as a read-modify-write. If the target cache block is not dirty, the operation completes in a single system bus transaction.

Address traces taken during product development were simulated to determine the frequency of dirty cache blocks that are targets of DMA writes. Our simulations showed that, for the address trace we used, frequency was extremely rare. Measurement taken from several applications and benchmarks confirmed that a dirty cache block is almost never asserted with a partial cache line write.

The DMA transfer of blocks that are aligned multiples of 16 bytes but less than a cache line is four times more efficient in the 4100 system than in earlier DIGITAL implementations.

Movement of blocks of less than 64 bytes is important to application performance because there are high-performance devices that move less than 64 bytes. One example is DIGITAL's MEMORY CHANNEL adapter, which moves 32-byte blocks in a burst.[2] As MEMORY CHANNEL adapters move large numbers of blocks that are all less than a cache line of data, the I/O subsystem partial cache line write feature improves system bus utilization and eliminates the system bus as a bottleneck. Message latency across the fabric of an AlphaServer 4100 MEMORY CHANNEL cluster (version 1.0) is approximately 6 microseconds (μs). There are two DMA writes in the message: the first is a message, and the second is a flag to validate the message. These DMA writes on the target AlphaServer 4100 contribute to message latency. The improvement in latency provided by the partial cache line write feature is approximately 0.5 μs per write. With two writes per message, latency is reduced by approximately 15 percent over an AlphaServer 4100 system with the partial cache line write feature. With version 1.5 of MEMORY CHANNEL adapters, net latency will improve by about 3 μs, and the effect of partial cache line writes will approach a 30 percent improvement in message latency.

In summary, the challenge is to efficiently move a block of data of a common size (multiple of 16 bytes) that is smaller than a cache line into the cache-coherent domain. Without any further improvement, the technique reduces system bus utilization by as much as a factor of four. This technique allows subblocks to be merged without incurring the overhead of read-modify-write, yet maintains cache coherency. The only drawback to the technique is some increased complexity in the CPU cache controller to support this mode. We considered the alternative of adding a small cache to the PCI bridge. Writes into the same memory region that occur within a short period of time could merge directly into a cache. This approach adds significant complexity and increases performance only if transactions that target the same cache line are very close together in time.

Peer-to-Peer Transaction Support

System bus and PCI bus utilization can be optimized for certain applications by limiting the number of times the same block of data moves through the system. As noted in the section AlphaServer 4100 System Overview, the PCI subsystem can contain two or four independent PCI bus bridges. Our design allows external devices connected to these separate peer PCI bus bridges to share data without accessing main memory and by using a minimal amount of host bus bandwidth. In other words, external devices can effect direct access to data on a peer-to-peer basis.

In conventional systems, a data file on a disk that is requested by a client node is transferred by DMA from the disk, across the PCI and the system bus, and into main memory. Once the data is in main memory, a network device can read the data directly in memory and send it across the network to the client node. In a 4100 system, device peer-to-peer transaction circumvents the transfer to main memory. However, peer-to-peer transaction requires that the target device have certain properties. The essential property is that the device target appear to the source device as if it is main memory.

The balance of this section explains how conventional DMA reads and writes are performed on the AlphaServer 4100 system, how the infrastructure for conventional DMA can be used for peer-to-peer transactions, and how deadlock avoidance is accomplished.

Conventional DMA

We extended the features of conventional DMA on the AlphaServer 4100 system to support peer-to-peer transaction. Conventional DMA in the 4100 system works as follows.

Address space on the Alpha processor is 2^{40} or 1 terabyte; the AlphaServer 4100 system supports up to 8 gigabytes (GB) of main memory. To directly address all of memory without using memory management hardware, an address must be 33 bits. (Eight GB is equivalent to 2^{33} bytes.)

Because the amount of memory is large compared to address space available on the PCI, some sort of memory management hardware and software is needed to make memory directly addressable by PCI devices. Most PCI devices use 32-bit DMA addresses. To provide direct access for every PCI device to all of the system address space, the PCI bus bridge has memory management hardware similar to that which is used on a CPU daughter card. Each PCI bridge to the system bus has a translation look-aside buffer (TLB) that converts PCI addresses into system bus addresses. The use of a TLB permits hardware to make all of physical memory visible through a relatively small region of address space that we call a DMA window.

A DMA window can be specified as "direct mapped" or "scatter-gather mapped." A direct-mapped DMA window adds an offset to the PCI address and passes it on to the system bus. A scatter-gather mapped DMA window uses the TLB to look up the system bus address.

Figure 3 is an example of how PCI memory address space might be allocated for DMA windows and for PCI device control status registers (CSRs) and memory.

A PCI device initiates a DMA write by driving an address on the bus. In Figure 4, data from PCI devices 0 and 1 are sent to the scatter-gather DMA windows; data from PCI device 2 are sent to the direct-mapped DMA window. When an address hits in one of the DMA windows, the PCI bus bridge acknowledges the address and immediately begins to accept write data. While consuming write data in a buffer, the PCI bus bridge translates the PCI address into a system address. The bridge then arbitrates for the system bus and, using the translated address, completes the write transaction. The write transaction completes on the PCI before it completes on the system bus.

A DMA read transaction has a longer latency than a DMA write because the PCI bus bridge must first translate the PCI address into a system bus address and fetch the data before completing the transaction. That is to say, the read transaction completes on the system bus before it can complete on the PCI.

Figure 5 shows the address path through the PCI bus bridge. All DMA writes and reads are ordered

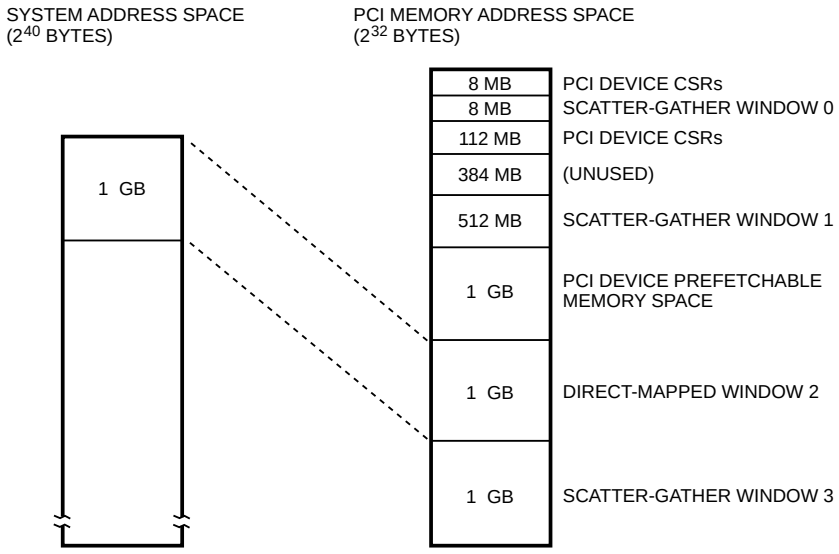


Figure 3
Example of PCI Memory Address Space Mapped to DMA Windows

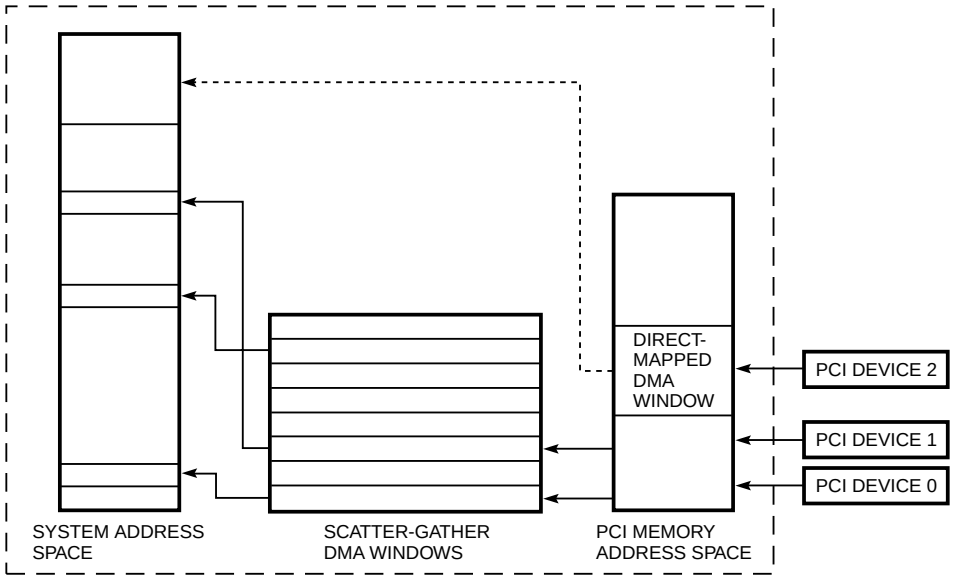


Figure 4
Example of PCI Device Reads or Writes to DMA Windows and Address Translation to System Bus Addresses

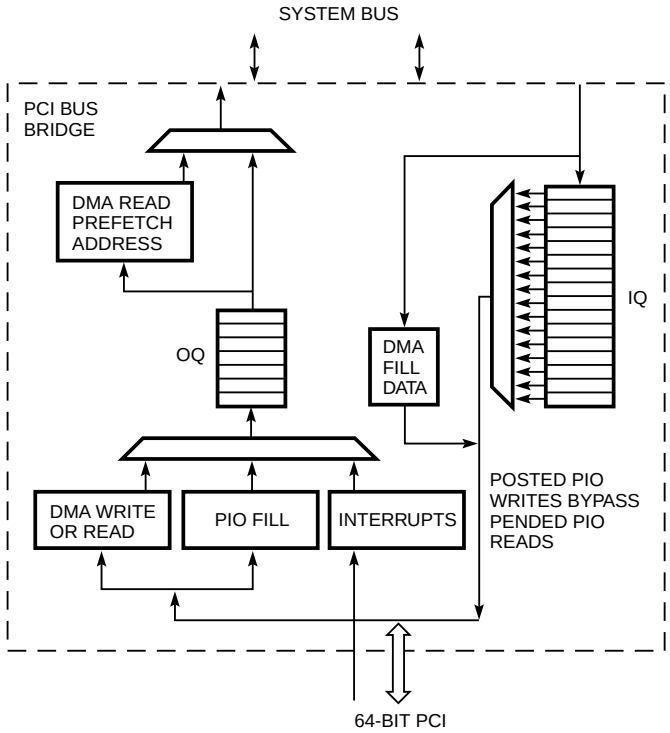


Figure 5
Diagram of Data Paths in a Single PCI Bus Bridge

through the outgoing queue (OQ) en route to the system bus. DMA read data is passed through an incoming queue (IQ) bypass by way of a DMA fill data buffer en route to the PCI.

Note that the IQ orders CPU-initiated PIO transactions. The IQ bypass is necessary for correct, deadlock-free operation of peer-to-peer transactions, which are explained in the next section.

Following is an example of how a conventional "bounce" DMA operation is used to move a file from a local storage device to a network device. The example illustrates how data is written into memory by one device where it is temporarily stored. Later the data is read by another DMA device. This operation is called a "bounce I/O" because the data "bounces" off

memory and out a network port, a common operation for a network file server application.

Assume PCI device A is a storage controller and PCI device B is a network device:

1. The storage controller, PCI device A, writes the file into a buffer on the PCI bus bridge using an address that hits a DMA window.
2. The PCI bridge translates the PCI memory address into a system bus address and writes the data into memory.
3. The CPU passes the network device a PCI memory space address that corresponds to the system bus address of the data in memory.
4. The network controller, PCI device B, reads the file in main memory using a DMA window and sends the data across the network.

If both controllers are on the same PCI bus segment and if the storage controller (PCI device A) could write directly to the network controller (PCI device B), no traffic would be introduced on the system bus. Traffic on the system bus is reduced by saving one DMA write, possibly one copy operation, and one DMA read. On the PCI bus, traffic is also reduced because there is one transaction rather than two. When the target of a transaction is a device other than main memory, the transaction is called a peer-to-peer. Peer-to-peer transactions on a single-bus system are simple, bordering on trivial; but deadlock-free support on a system with multiple peer PCI buses is quite a bit more difficult.

This section has presented a high-level description of how a PCI device DMA address is translated into a system bus address and data are moved to or from main memory. In the next section, we show how the same mechanism is used to support device peer-to-peer transactions and how traffic is managed for deadlock avoidance.

A Peer-to-Peer Link Mechanism

For direct peer-to-peer transactions to work, the target device must behave as if it is main memory; that is, it must have a target address in prefetchable PCI memory space.[4] The PCI specification further states that devices are not allowed to depend on completion of a transaction as master.[5] Two devices supported by the DIGITAL UNIX operating system meet these criteria today with some restrictions; these are the MEMORY CHANNEL adapter noted earlier and the Prestoserve NVRAM, a nonvolatile memory storage device used as an accelerator for transaction processing. The PNVRAM was part of the configuration in which the AIM benchmark results cited in the introduction were achieved.

Both conventional DMA and peer-to-peer transactions work the same way from the perspective of the PCI master: The device driver provides the master device with a target address, size of the transfer, and identification of data to be moved. In the case in which a data file is to be read from a disk, the device driver software gives the PCI device that controls the disk a "handle," which is an identifier for the data file and the PCI target address to which the file should be written. To reiterate, in a conventional DMA transaction, the target address is in one of the PCI bus bridge DMA windows. The DMA window logic translates the address into a main memory address on the system bus. In a peer-to-peer transaction, the target address is translated to an address assigned to another PCI device.

Any PCI device capable of DMA can perform peer-to-peer transactions on the AlphaServer 4100 system. For example, in Figure 6, PCI device A can transfer data to or from PCI device B without using any resources or facilities in the system bus bridge. The use of a peer-to-peer transaction is controlled entirely by software: The device driver passes a target address to PCI device A, and device A uses the address as the DMA data source or destination.

If the target of the transaction is PCI device C, then system services software allocates a region in a scatter-gather map and specifies a translation that maps the scatter-gather-mapped address on PCI bus 0 to a system bus address that maps to PCI device C. This address translation is placed in the scatter-gather map. When PCI device A initiates a transaction, the address matches one of the DMA windows that has been initialized for scatter-gather. The PCI bus bridge accepts the transaction, looks up the translation in the scatter-gather map, and uses a system address that maps through PCI bus bridge 1 to hit PCI device C. The transaction on the system bus is between the two PCI bridges, with no involvement by memory or CPUs. In this transaction, the system bus is utilized, but the data is not stored in main memory. This eliminates the intermediate steps and overhead associated with conventional DMA, traditionally done by the "bounce" of the data through main memory.

The features that allow software to make a device on one PCI bus segment visible to a device on another are all implicit in the scatter-gather mapping TLB. For peer-to-peer transaction support, we extended the range of translated addresses to include memory space on peer PCI buses. This allows address space on one independent PCI bus segment to appear in a window of address space on a second independent peer PCI bus segment. On the system bus, the peer transaction hits in the address space of the other PCI bridge.

Deadlock Avoidance in Device Peer-to-Peer Transactions

The definition of deadlock, as it is solved in this design, is the state in which no progress can be made on any transaction across a bridge because the queues are filled with transactions that will never complete.

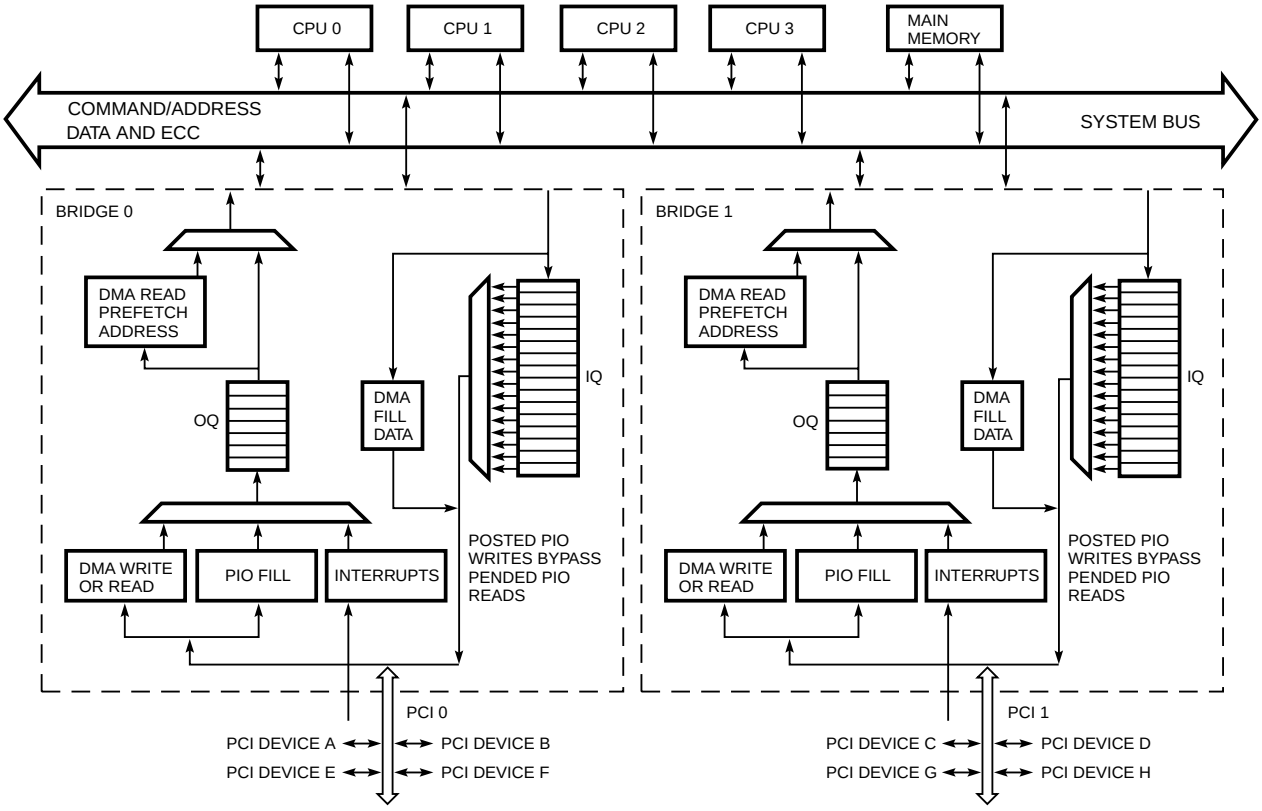


Figure 6
AlphaServer 4100 System Diagram Showing Data Paths through PCI Bus Bridges

A deadlock situation is analogous to highway gridlock in which two lines of automobiles face each other on a single-lane road; there is no room to pass and no way to back up. Rules for deadlock avoidance are analogous to the rules for directing vehicle traffic on a narrow bridge.

An example of peer-to-peer deadlock is one in which two PCI devices are dependent on the completion of a write as masters before they will accept writes as targets. When these two devices target one another, the result is deadlock; each device responds with RETRY to every write in which it is the target, and each device is unable to complete its current write transaction because it is being retried.

A device that does *not* depend on completion of a transaction as master before accepting a transaction as target may also cause deadlocks in a bridged environment. Situations can occur on a bridge in which multiple outstanding posted transactions must be kept in order. Careful design is required to avoid the potential for deadlock.

The design for deadlock-free peer-to-peer transaction support in the AlphaServer 4100 system includes the

- Implementation of PCI delayed-read transactions
- Use of bypass paths in the IQ and in read-return data

This section assumes that the reader is familiar with the PCI protocol and ordering rules.[4]

Figure 6 shows the data paths through two PCI bus bridges. Transactions pass through these bridges as follows:

- CPU software-initiated PIO reads and PIO writes are entries in the IQ.
- Device peer-to-peer transactions targeting devices on peer PCI segments also use the IQ.
- PCI-device–initiated reads and writes (DMA or peer-to-peer), interrupts, and PIO fill data are entries in the OQ.
- The multiplexer selecting entries in the IQ allows writes (PIO or peer-to-peer) to bypass delayed (pended) reads (PIO or peer-to-peer).
- The read prefetch address register permits read-return in the OQ data to bypass PCI delayed reads.

The two bypass paths around the IQ and OQ are required to avoid deadlocks that may occur during device peer-to-peer transactions. All PCI ordering rules are satisfied from the point of view of any single device in the system. The following example demonstrates deadlock avoidance in a device peer-to-peer write and a device peer-to-peer read, referencing Figure 7.

The configuration in the example is an AlphaServer 4100 system with four CPUs and two PCI bus bridges. Devices A and C are simple master-capable DMA controllers, and devices B and D are simple targets, e.g., video RAMs, network controllers, PNVRAM, or any device with prefetchable memory as defined in the PCI standard.

Example of device peer-to-peer write block completion of pended PIO read-return data:

1. PCI device A initiates a peer-to-peer burst write targeting PCI device D.
2. Write data enters the OQ on bridge 0, filling three posted write buffers.
3. The target bridge, bridge 1, writes data from bridge 0.
4. When the IQ on bridge 1 hits a threshold, it uses the system bus flow-control to hold off the next write.
5. As each 64-byte block of write data is retired out of the IQ on bridge 1, an additional 64-byte (cache line size) write of data is allowed to move from the OQ on bridge 0 to the IQ on bridge 1.
6. If the OQ on bridge 0 is full, bridge 0 will disconnect from the current PCI transaction and will retry all transactions on PCI 0 until an OQ slot becomes available.
7. PCI device C initiates a peer-to-peer burst write, targeting PCI device B; the same scenario follows as steps 1 through 6 above but in the opposite direction.
8. CPU 0 posts a read of PCI memory space on PCI device E.
9. CPU 1 posts a read of PCI memory space on PCI device G.
10. CPU 2 posts a read of PCI memory space on PCI device F.
11. CPU 3 posts a read of PCI memory space on PCI device H.
12. Deadlock:
    - Both OQs are stalled waiting for the corresponding IQ to complete an earlier posted write.
    - The design has two PIO read-return data (fill) buffers; each is full.
    - The PIO read-return data must stay behind the posted writes to satisfy PCI-specified posted write buffer flushing rules.
    - A third read is at the bottom of each IQ, and it cannot complete because there is no fill buffer available in which to put the data.

To avoid this deadlock, posted writes are allowed to bypass delayed (pended) reads in the IQ, as shown in Figure 6. In the AlphaServer 4100 deadlock-avoidance design, the IQ will always empty, which in turn allows the OQ to empty.

Note that the IQ bypass logic implemented for deadlock avoidance on the AlphaServer 4100 system may appear to violate General Rule 5 from the PCI specification, Appendix E:

> A read transaction must push ahead of it through the bridge any posted writes originating on the same side of the bridge and posted before the read. Before the read transaction can complete on its originating bus, it must pull out of the bridge any posted writes that originated on the opposite side and were posted before the read command completes on the read-destination bus.[4]

In fact, because of the characteristics of the CPUs and the flow-control mechanism on the system bus, all rules are followed as observed from any single CPU or PCI device in the system. Because reads that target a PCI address are always split into separate request and response transactions, the appropriate ordering rule for this case is PCI Specification Delayed Transaction Rule 7 in Section 3.3.3.3 of the PCI specification:

> Delayed Requests and Delayed Completions have no ordering requirements with respect to themselves or each other. Only a Delayed Write Completion can pass a Posted Memory Write. A Posted Memory Write must be given an opportunity to pass everything except another Posted Memory Write.[4]

Also note that, as shown in Figure 6, the DMA fill data buffers bypass the IQ, apparently violating General Rule 5. The purpose of General Rule 5 is to provide a mechanism in a device on one side of a bridge to ensure that all posted writes have completed. This rule is required because interrupts on PCI are side-band signals that may bypass all posted data and signal completion of a transaction before the transaction has actually completed. In the AlphaServer 4100 system, all writes to or from PCI devices are strictly ordered, and there is no side-band signal notifying a PCI device of an event. These system characteristics allow the PCI bus bridge to permit DMA fill data (in PCI lexicon, this could be a delayed-read completion, or read data in a connected transaction) to bypass posted memory writes in the IQ. This bypass is necessary to limit PCI target latency on DMA read transactions.

We have presented two IQ bypass paths in the AlphaServer 4100 design. We describe one IQ bypass as a required feature for deadlock avoidance in peer-to-peer transactions between devices on different buses. The second bypass is required for performance reasons and is discussed in the section I/O Bandwidth and Efficiency.

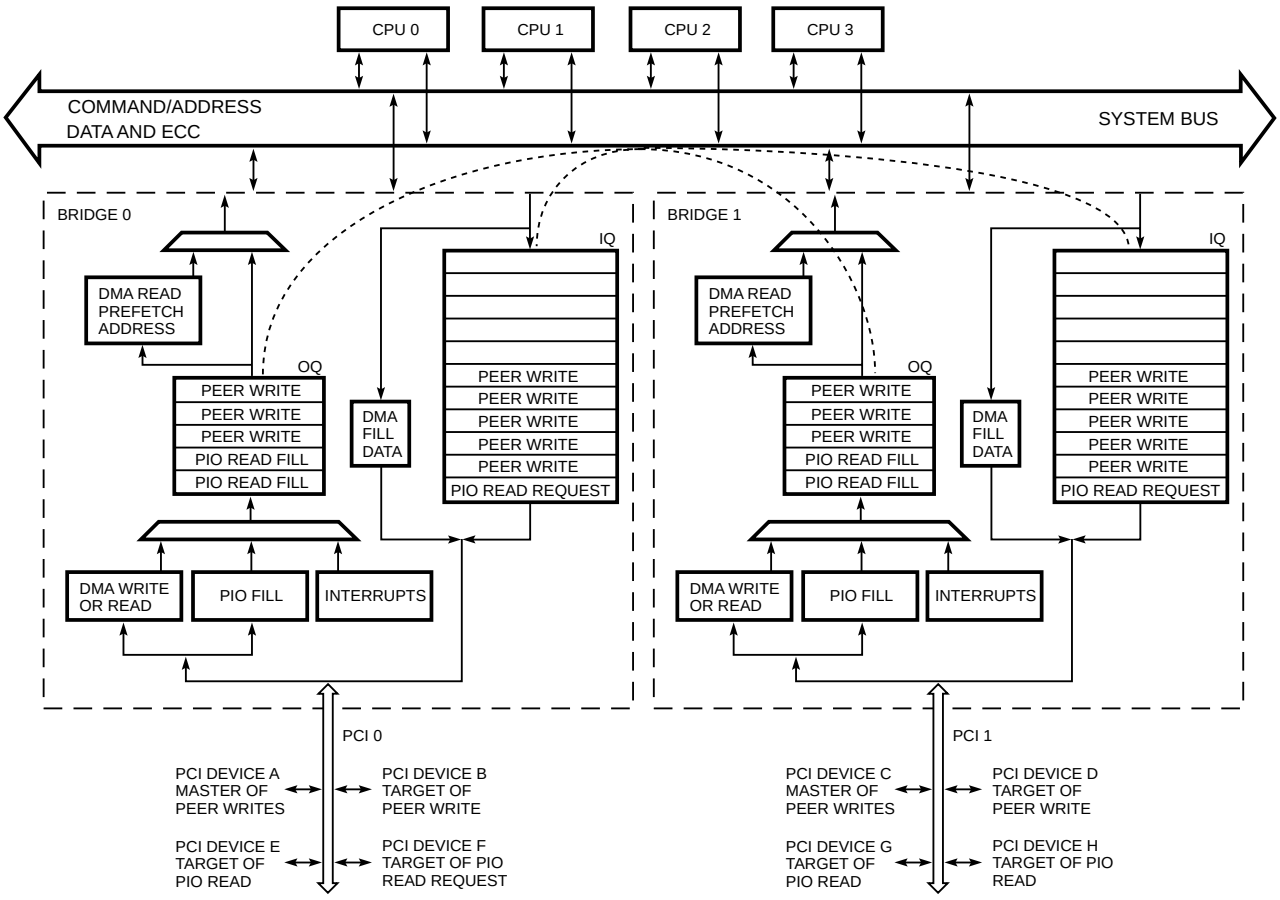


Figure 7
Block Diagram Showing Deadlock Case without IQ Bypass Path

Required Characteristics for Deadlock-free Peer-to-Peer Target Devices

PCI devices must follow all PCI standard ordering rules for deadlock-free peer-to-peer transaction. The specific rule relevant to the AlphaServer 4100 design for peer-to-peer transaction support is Delayed Transaction Rule 6, which guarantees that the IQ will always empty:

> A target must accept all memory writes addressed to it while completing a request using Delayed Transaction termination.[4]

Our design includes a link mechanism using scatter-gather TLBs to create a logical connection between two PCI devices. It includes a set of rules for bypassing data that ensures deadlock-free operation when all participants in a peer-to-peer transaction follow the ordering rules in the PCI standard. The link mechanism provides a logical path for peer-to-peer transactions and the bypassing rules guarantee the IQ will always drain. The key feature, then, is a guarantee that the IQ will always drain, thus ensuring deadlock-free operation.

I/O Bandwidth and Efficiency

With overall system performance as our goal, we selected two design approaches to deliver full PCI bandwidth without bus stalls. These were support for large bursts of PCI-device-initiated DMA, and sufficient buffering and prefetching logic to keep up with the PCI and avoid introducing stalls. We open this section with a review of the bandwidth and latency issues we examined in our efforts to achieve greater bandwidth efficiency.

The bandwidth available on a platform is dependent on the efficiency of the design and on the type of transactions performed. Bandwidth is measured in millions of bytes per second (MB/s). On a 32-bit PCI, the available bandwidth is efficiency multiplied by 133 MB/s; on a 64-bit PCI, available bandwidth is efficiency multiplied by 266 MB/s. By efficiency, we mean the amount of time spent actually transferring data as compared with total transaction time.

Both parties in a transaction contribute to efficiency on the bus. The AlphaServer 4100 I/O design keeps the overhead introduced by the system to a minimum and supports large burst sizes over which the per-transaction overhead can be amortized.

Support for Large Burst Sizes

To predict the efficiency of a given design, one must break a transaction into its constituent parts. For example, when an I/O device initiates a transaction it must

- Arbitrate for the bus
- Connect to the bus (by driving the address of the transaction target)
- Transfer data (one or more bytes move in one or more bus cycles)
- Disconnect from the bus

Time actually spent in an I/O transaction is the sum of arbitration, connection, data transfer, and disconnection.

The period of time before any data is transferred is typically called latency. With small burst sizes, bandwidth is limited regardless of latency. Latency of arbitration, connection, and disconnection is fairly constant, but the amount of data moved per unit of time can increase by making the I/O bus wider. The AlphaServer 4100 PCI buses are 64 bits wide, yielding (efficiency × 266 MB/s) of available bandwidth.

As shown in Figure 8, efficiency improves as burst size increases and overhead (i.e., latency plus stall time) decreases. Overhead introduced by the AlphaServer 4100 is fairly constant. As discussed earlier, a DMA write can complete on the PCI before it completes on the system bus. As a consequence, we were able to keep overhead introduced by the platform to a minimum for DMA writes. Recognizing that efficiency improves with burst size, we used a queuing model of the system to predict how many posted write buffers were needed to sustain DMA write bursts without stalling the PCI bus. Based on a simulation model of the configurations shown in Figures 1 and 2, we determined that three 64-byte buffers were sufficient to stream DMA writes from the (266 MB/s) PCI bus to the (1 GB/s) system bus.

Later in this paper, we present measured performance of DMA write bandwidth that matches the simulation model results and, with large burst sizes, actually exceeds 95 percent efficiency.

Prefetch Logic

DMA writes complete on the PCI before they complete on the system bus, but DMA reads must wait for data fetched from memory or from a peer on another PCI. As such, latency for DMA reads is always worse than it is for writes. *PCI Local Bus Specification Revision 2.1* provides a delayed-transaction mechanism for devices with latencies that exceed the PCI initial-latency requirement.[4] The initial-latency requirement on host bus bridges is 32 PCI cycles, which is the maximum overhead that may be introduced before the first data cycle. The AlphaServer 4100 initial latency for memory DMA reads is between 18 and 20 PCI

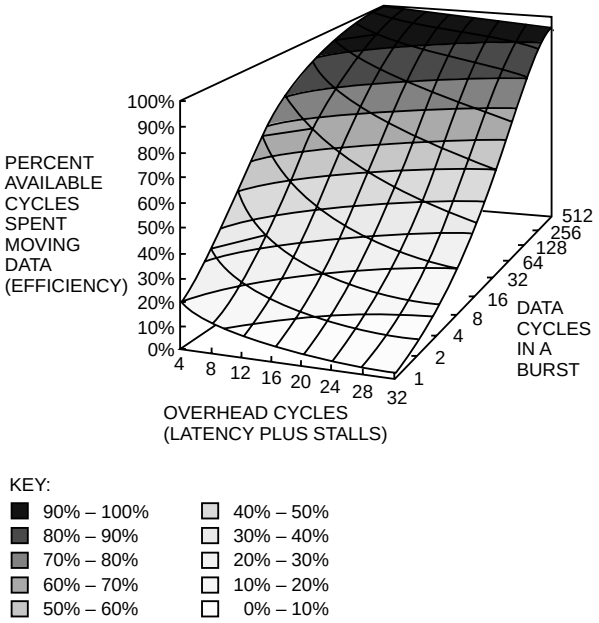


Figure 8
PCI Efficiency as a Function of Burst Size and Latency

cycles. Peer-to-peer reads of devices on different bus segments are always converted to delayed-read transactions because the best-case initial latency will be longer than 32 PCI cycles.

PCI initial latency for DMA reads on the AlphaServer 4100 system is commensurate with expectations for current generation quad-processor SMP systems. To maximize efficiency, we designed prefetching logic to stream data to a 64-bit PCI device without stalls after the initial-latency penalty has been paid. To make sure the design could keep up with an uninterrupted 64-bit DMA read, we used the queuing model and analysis of the system bus protocol and decided that three cache-line-size prefetch buffers would be sufficient. The algorithm for prefetching uses the advanced PCI commands as hints to determine how far memory data prefetching should stay ahead of the PCI bus:

- Memory Read (MR): Fetch a single 64-byte cache line.
- Memory Read Line (MRL): Fetch two 64-byte cache lines.
- Memory Read Multiple (MRM): Fetch two 64-byte cache lines, and then fetch one line at a time to keep the pipeline full.

After the PCI bus bridge responds to an MRM command by fetching two 64-byte cache lines and the second line is returned, the bridge posts another read; as the oldest buffer is unloaded, new reads are posted, keeping one buffer ahead of the PCI. The third prefetch buffer is reserved for the case in which a DMA

MRM completes while there are still prefetch reads outstanding. Reservation of this buffer accomplishes two things: (1) it eliminates a time-delay bubble that would appear between consecutive DMA read transactions, and (2) it maintains a resource to fetch a scatter-gather translation in the event that the next transaction address is not in the TLB. Measured DMA bandwidth is presented later in this paper.

The point at which the design stops prefetching is on page boundaries. As the DMA window scatter-gather map is partitioned into 8-KB pages, the interface is designed to disconnect on 8-KB–aligned addresses.

The advantage of prefetching reads and absorbing posted writes on this system is that the burst size can be as large as 8 KB. With large burst size, the overhead of connecting and disconnecting from the bus is amortized and approaches a negligible penalty.

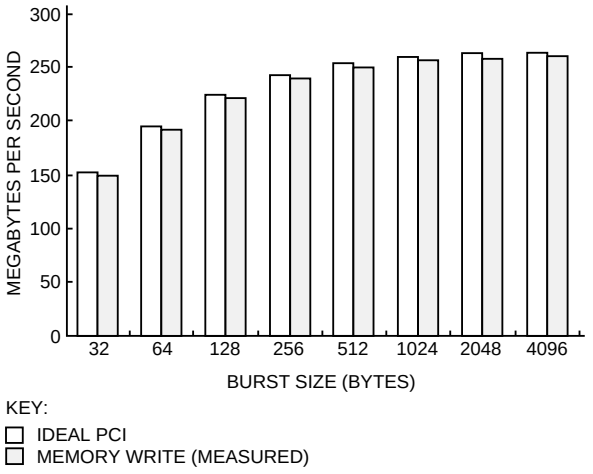


Figure 9
Comparison of Measured DMA Write Performance on an Ideal 64-bit PCI and on an AlphaServer 4100 System

DMA and PIO Performance Results

We have discussed the relationship between burst size, initial latency, and bandwidth and described several techniques we used in the AlphaServer 4100 PCI bus bridge design to meet the goals for high-bandwidth I/O. This section presents the performance delivered by the 4100 I/O subsystem design, which has been measured using a high-performance PCI transaction generator.

We collected performance data under the UNIX operating system with a reconfigurable interface card developed at DIGITAL, called PCI Pamette. It is a 64-bit PCI option with a Xilinx FPGA interface to PCI. The board was configured as a programmable PCI transaction generator. In this configuration, the board can generate burst lengths of 1 to 512 cycles. DMA either runs to a fixed count of words transferred or runs continuously (software selected). The DMA engine runs at a fixed cadence (delay between bursts) of 0 to 15 cycles in the case of a fixed count and at 0 to 63 cycles when run continuously.

The source of the DMA is a combination of a free-running counter that is clocked using the PCI clock and a PCI transaction count. The free-running counter time-stamps successive words and detects wait states and delays between transactions. The transaction count identifies retries as well as transaction boundaries.

As the target of PIO read or write, the board can accept arbitrarily large bursts of either 32 or 64 bits. It is a medium decode device and always operates with zero wait states.

DMA Write Efficiency and Performance

Figure 9 shows the close comparison between the AlphaServer 4100 system and a nearly perfect PCI design in measured DMA write bandwidth. As explained above, to sustain large bursts of DMA writes, we implemented three 64-byte posted write buffers. Simulation predicted that this number of buffers would be sufficient to sustain full bandwidth DMA writes—even when the system bus is extremely busy—because the bridges to the PCI are on a shared system bus that has roughly 1 GB/s available bandwidth. The PCI bus bridges arbitrate for the shared system bus at a priority higher than the CPUs, but the bridges are permitted to execute only a single transaction each time they win the system bus. Therefore, in the worst case, a PCI bus bridge will wait behind three other PCI bus bridges for a slot on the bus, and each bridge will have at least one quarter of the available system bus bandwidth. With 250 MB/s available but with potential delay in accessing the bus, three posted write buffers are sufficient to maintain full PCI bandwidth for memory writes.

The ideal PCI system is represented by calculated performance data for comparison purposes. It is a system that has three cycles of target latency for writes. Three cycles is the best possible for a medium decode device. The goal for DMA writes was to deliver performance limited only by the PCI device itself, and this goal was achieved. Figure 9 demonstrates that measured DMA write performance on the AlphaServer 4100 system approaches theoretical maximums. The combination of optimizations and innovations used on this platform yielded an implementation that meets the goal for DMA writes.

DMA Read Efficiency and Performance

As noted in the section Prefetch Logic, bandwidth performance of DMA reads will be lower than the performance of DMA writes on all systems because there is delay in fetching the read data from memory. For this reason, we included three cache-line–size prefetch buffers in the design.

Figure 10 compares DMA read bandwidth measured on the AlphaServer 4100 system with a PCI system that has 8 cycles of initial latency in delivering DMA read data. This figure shows that delivered bandwidth improves on the AlphaServer 4100 system as burst size increases, and that the effect of initial latency on measured performance is diminished with larger DMA bursts.

The ideal PCI system used calculated performance data for comparison, assuming a read target latency of 8 cycles; 2 cycles are for medium decode of the address, and 6 cycles are for memory latency of 180 nanoseconds (ns). This represents about the best performance that can be achieved today.

Figure 10 shows memory read and memory read line commands with burst sizes limited to what is expected from these commands. As explained elsewhere in this paper, *memory read* is used for bursts of less than a cache line; *memory read line* is used for transactions that cross one cache line boundary but are less than two cache lines; and *memory read multiple* is for transactions that cross two or more cache line boundaries.

The efficiency of *memory read* and *memory read line* does not improve with larger bursts because there is no prefetching beyond the first or second cache line respectively. This shows that large bursts and use of the appropriate PCI commands are both necessary for efficiency.

Performance of PIO Operations

PIO transactions are initiated by a CPU. AlphaServer 4100 PIO performance has been measured on a system with a single CPU, and the results are presented in Figure 11. The pended protocol for flow control on the system bus limits the number of read transactions that can be outstanding from a single CPU. A single CPU issuing reads will stall waiting for read-return data and cannot issue enough reads to approach the bandwidth limit of the bridge. Measured read performance is quite a bit lower than the theoretical limit. A system with multiple CPUs doing PIO reads—or peer-to-peer reads—will deliver PIO read bandwidth that approaches the predicted performance of the PCI bus bridge. PIO writes are posted and the CPU stalls only when the writes reach the IQ threshold. Figure 11 shows that PIO writes approach the theoretical limit of the host bus bridge.

PIO bursts are limited by the size of the I/O read and write merge buffers on the CPU. A single AlphaServer 4100 CPU is capable of bursts up to 32 bytes. PIO writes are posted; therefore, to avoid stalling the system with system bus flow control, in the maximum configuration (see Figure 2), we provide a minimum of three posted write buffers that may be filled before flow control is used. Configurations with fewer than the maximum number of CPUs can post more PIO writes before encountering flow control.

Summary

The DIGITAL AlphaServer 4100 system incorporates design innovations in the PCI bus bridge that provide a highly efficient interface to I/O devices. Partial cache line writes improve the efficiency of small writes to memory. The peer link mechanism uses TLBs to

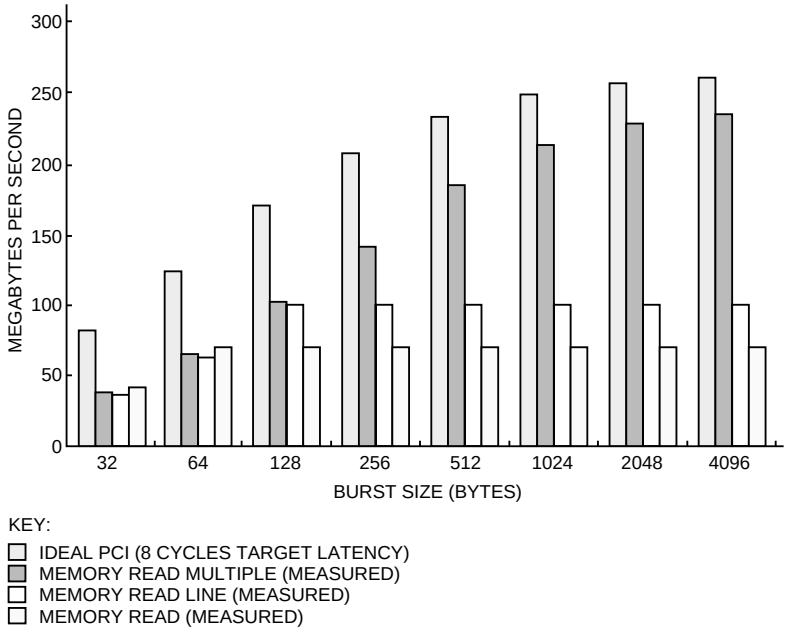


Figure 10
Comparison of DMA Read Bandwidth on the AlphaServer 4100 System and on an Ideal PCI System

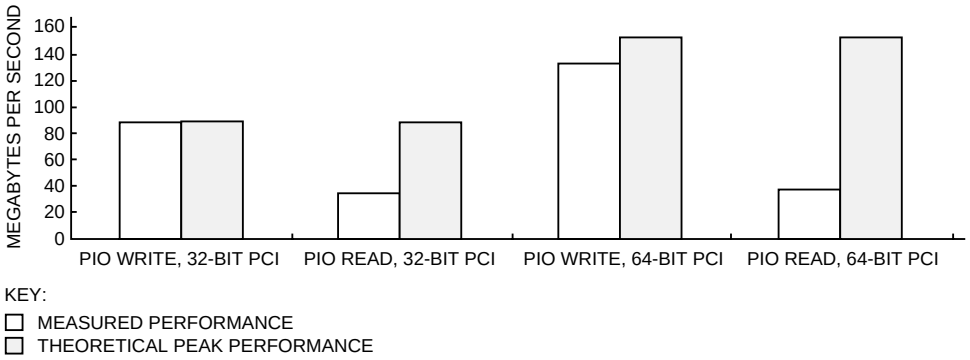


Figure 11
Comparison of AlphaServer 4100 PIO Performance with Theoretical 32-byte Burst Peak Performance

map device address space on independent peer PCI buses to permit direct peer transactions. Reordering of transactions in queues on the PCI bridge, combined with the use of PCI delayed transactions, provides a deadlock-free design for peer transactions. Buffers and prefetch logic that support very large bursts without stalls yield a system that can amortize overhead and deliver performance limited only by the PCI devices used in the system.

In summary, this system meets and exceeds the performance goals established for the I/O subsystem. Notably, I/O subsystem support for partial cache line writes and for direct peer-to-peer transactions significantly improves efficiency of operation in a MEMORY CHANNEL cluster system.

Acknowledgments

The DIGITAL AlphaServer 4100 I/O design team was responsible for the I/O subsystem implementation. The design team included Bill Bruce, Steve Coe, Dennis Hayes, Craig Keefer, Andy Koning, Tom McLaughlin, and John Lynch. The I/O design verification team was also key to delivering this product: Dick Beaven, Dmetro Kormeluk, Art Singer, and Hitesh Vyas, with CAD support from Mark Matulatis and Dick Lombard.

Several system team members contributed to inventions that improved product performance; most notable were Paul Guglielmi, Rick Hetherington, Glen Herdeg, and Maurice Steinman. We also extend thanks to our performance partners Zarka Cvetanovic and Susan Carr, who developed and ran the queuing models.

Mark Shand designed the PCI Pamette and provided the performance measurements used in this paper. Many thanks for the nights and weekends spent remotely connected to the system in our lab to gather this data.

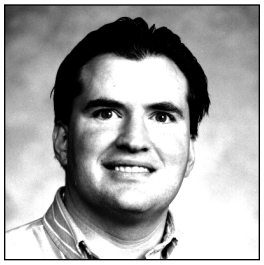
References and Note

1. Winter UNIX Hot Iron Awards, UNIX EXPO Plus, October 9, 1996, http://www.aim.com (Menlo Park, Calif.: AIM Technology).
2. R. Gillett, "MEMORY CHANNEL Network for PCI," *IEEE Micro* (February 1996): 12–18.
3. G. Herdeg, "Design and Implementation of the AlphaServer 4100 CPU and Memory Architecture," *Digital Technical Journal,* vol. 8, no. 4 (1996, this issue): 48–60.
4. *PCI Local Bus Specification, Revision 2.1* (Portland, Oreg.: PCI Special Interest Group, 1995).
5. In PCI terminology, a master is any device that arbitrates for the bus and initiates transactions on the PCI (i.e., performs DMA) before accepting a transaction as target.

Biographies

Samuel H. Duncan
A consultant engineer and the architect for the AlphaServer 4100 I/O subsystem design, Sam Duncan is currently working on core logic design and architecture for the next generation of Alpha servers and workstations. Since joining DIGITAL in 1979, he has been part of Alpha and VAX system engineering teams and has represented DIGITAL on several industry standards bodies, including the PCI Special Interest Group. He also chaired the group that developed the IEEE Standard for Communicating Among Processors and Peripherals Using Shared Memory. He has been awarded one patent and has four patents filed for inventions in the AlphaServer 4100 system. Sam received a B.S.E.E. from Tufts University.

Craig D. Keefer
Craig Keefer is a principal hardware engineer whose engineering expertise is designing gate arrays. He was the gate array designer for one of the two 235K CMOS gate arrays in the AlphaServer 8200 system and the team leader for the command and address gate array in the AlphaServer 8400 I/O module. A member of the Server Product Development Group, he is now responsible for designing gate arrays for hierarchical switch hubs. Craig joined DIGITAL in 1977 and holds a B.S.E.E from the University of Lowell.



Thomas A. McLaughlin
Tom McLaughlin is a principal hardware engineer working in DIGITAL's Server Product Development Group. He is currently involved with the next generation of high-end server platforms and is focusing on logic synthesis and ASIC design processes. For the AlphaServer 4100 project, he was responsible for the logic design of the I/O subsystem, including ASIC design, logic synthesis, logic verification, and timing verification. Prior to joining the AlphaServer 4100 project, he was a member of Design and Applications Engineering within DIGITAL's External Semiconductor Technology Group. Tom joined DIGITAL in 1986 after receiving a B.T.E.E.T. from the Rochester Institute of Technology; he also holds an M.S.C.S. degree from the Worcester Polytechnic Institute.