The AlphaServer 2100 I/O Subsystem

by

Andrew P. Russo

ABSTRACT

The AlphaServer 2100 I/O subsystem contains a dual-level I/O structure that includes the high-powered PCI local bus and the widely used EISA bus. The PCI bus is connected to the server's multiprocessing system bus through the custom-designed bridge chip. The EISA bus supports eight general-purpose EISA/ISA connectors, providing connections to plug-in, industry-standard options. Data rate isolation, disconnected transaction, and data buffer management techniques were used to ensure bus efficiency in the I/O subsystem. Innovative engineering designs accomplished the task of combining Alpha CPUs and standard-system I/O devices.

INTRODUCTION

Digital's AlphaServer 2100 server combines Alpha multiprocessing technology with an I/O subsystem typically associated with personal computers (PCs).[1] The I/O subsystem on the AlphaServer 2100 system contains a two-level hierarchical bus structure consisting of a high-performance primary I/O bus connected to a secondary, lower performance I/O bus. The primary I/O bus is a 32-bit peripheral component interconnect (PCI) local bus (or simply, PCI bus).[2] The PCI bus is connected to the AlphaServer 2100 system's multiprocessing system bus through a custom application specific integrated circuit (ASIC) bridge chip (referred to as the T2 bridge chip). The secondary I/O bus is a 32-bit Extended Industry Standard Architecture (EISA) bus connected to the PCI bus through a bridge chip set provided by Intel Corporation.[3] Figure 1 shows the I/O subsystem designed for the AlphaServer 2100 product. The I/O subsystem demonstrated sufficient flexibility to become the I/O interface for the small pedestal AlphaServer 2000 product and the rackmountable version of the AlphaServer 2100 server.

[Figure 1 (I/O Subsystem for the AlphaServer 2100 System)
is not available in ASCII format.]

This paper discusses the dual-level bus hierarchy and the several I/O advantages it provides. The design considerations of the I/O subsystem for the AlphaServer 2100 server are examined in the sections that follow.

I/O SUPPORT FOR EISA AND PCI BUSES

The EISA bus enables the AlphaServer 2100 system to support a wide range of existing EISA or Industry Standard Architecture (ISA) I/O peripherals.[4] The EISA bus can sustain data rates up to a theoretical limit of 33 megabytes per second (MB/s) at a clock rate of 8.25 megahertz (MHz). In the current configuration for the AlphaServer 2100 product, the EISA bus supports eight general-purpose EISA/ISA connectors, and the EISA bridge chip set provides connections to various low-speed, system-standard I/O devices such as keyboard, mouse, and time-of-year (TOY) clock. For most system configurations, the AlphaServer 2100 system's EISA bus provides enough data bandwidth to meet all data throughput requirements. In light of the new requirements for faster data rates, however, the EISA bus will soon begin to run out of bus bandwidth.

To provide for more bandwidth, the AlphaServer 2100 system also contains a PCI bus as its primary bus. With data rates four times that of the EISA bus, the PCI bus provides a direct migration path from the EISA bus. The 32-bit PCI bus can sustain data rates up to a theoretical limit of 132 MB/s at a clock rate of 33 MHz. In the AlphaServer 2100 system configuration, the PCI bus provides connections to three general-purpose 32-bit PCI connectors, an Ethernet device, a SCSI device, the PCI-to-EISA bridge chip, and the T2 bridge chip.

A close examination of the bus structure reveals that the AlphaServer 2100 system actually contains a three-level, hierarchical bus structure. In addition to the PCI and EISA buses, the AlphaServer 2100 system includes a 128-bit multiprocessing system bus, as shown in Figure 1. Each bus is designed to adhere to its own bus interface protocols at different data rates. The system bus is 128 bits per 24 nanoseconds (ns); the PCI bus is 32 bits per 30 ns; and the EISA bus is 32 bits per 120 ns. Each bus is required to provide a particular function to the system and is positioned in the bus hierarchy to maximize that efficiency. For example, the system bus is positioned close to the CPUs and memory to maximize CPU memory access time, and the lower performance I/O devices are placed on the EISA bus because their timing requirements are less critical. To maintain maximum bus efficiency on all three buses, it is critical that each bus be able to perform its various functions autonomously of each other. In other words, a slower performing bus should not affect the efficiency of a high-performance bus. The section below discusses a few techniques that we designed into the I/O subsystem to enable the buses to work together efficiently.

USING THE BUS HIERARCHY EFFICIENTLY

This section discusses the data rate isolation, disconnected transaction, data buffer management, and data bursting techniques used to ensure bus efficiency in the I/O subsystem.

Data Rate Isolation

The three-level bus hierarchy promotes data rate isolation and concurrency for simultaneous operations on all three buses. The design of the bus bridges helps to enable each bus to work independently: it provides bus interfaces with extensive data buffering that function at the same data rates as the interfacing bus. For example, the T2 bridge chip contains both a system bus interface and a PCI bus interface that run synchronously to their respective buses, but are totally asynchronous to each other. The data buffers inside the T2 bridge chip act as a domain connector from one bus time zone to the other and help to isolate the data rates of the two buses.

Disconnected Transactions

Whenever possible, the bridges promote the use of disconnected (or pended) protocols to move data across the buses. Disconnected protocols decrease the interdependencies between the different buses. For example, when a CPU residing on the system bus needs to move data to the PCI bus, the CPU does so by sending its data onto the system bus. Here the T2 bridge chip (see Figure 2) stores the data into its internal data buffers at the system bus data rate. The T2 bridge chip provides enough buffering to store an entire CPU transaction. From the CPU's perspective, the transaction is completed as soon as the T2 bridge chip accepts its data. At that point, the T2 bridge chip must forward the data to the PCI bus, independent of the CPU. In this way, the CPU is not required to waste bus bandwidth by waiting for the transfer to complete to its final destination on the PCI bus.

[Figure 2 (Block Diagram of the T2 Bridge Chip) is not available in ASCII format.]

The T2 bridge chip implements disconnected transactions for all CPU-to-PCI transactions and most PCI-to-memory transactions. In a similar fashion, the PCI-to-EISA bridge implements disconnected transactions between the PCI bus and the EISA bus.

Data Buffer Management

In addition to containing temporary data buffering to store data on its journey from bus to bus, each bridge chip utilizes buffer management to allocate and deallocate its internal data buffers from one incoming data stream to another. In this way, a single ASIC bridge design can efficiently service multiple data streams with a relatively small amount of data buffering and without impacting bus performance.

The T2 bridge chip contains 160 bytes of temporary data buffering divided across the three specific bus transactions it performs.

These three transactions are (1) direct memory access (DMA) writes from PCI to memory (system bus), (2) DMA reads from memory (system bus) to PCI, and 3) programmed I/O (system bus) reads/writes by a CPU from/to the PCI. The T2 bridge chip's data buffering is organized into five 32-byte buffers. Two 32-byte buffers each are allocated to the DMA write and DMA read functions, and one 32-byte buffer is allocated to the programmed I/O function. Each of the three transaction functions contains its own buffer management logic to determine the best use of its available data buffering. Buffer management is especially valuable in situations in which a PCI device is reading data from memory on the system bus. To maintain an even flow of data from bus to bus, the buffer management inside the T2 bridge chip attempts to prefetch more read data from memory while it is moving data onto the PCI.

Buffer management helps the bridges service bus transactions in a way that promotes continuous data flow that, in turn, promotes bus efficiency.

Burst Transactions

Using a bus efficiently also means utilizing as much of the bus bandwidth as possible for "useful" data movement. Useful data movement is defined as that section of time when only the actual data is moving on the bus, devoid of address or protocol cycles. Maximizing useful data movement can be accomplished by sending many data beats (data per cycle) per single transfer time. Sending multiple data beats per single transfer is referred to as a "burst transaction."

All three buses have the ability to perform burst transactions. The system bus can burst as much as 32 bytes of data per transaction, and the PCI and EISA buses can burst continuously as required.

Data bursting promotes bus efficiency and very high data rates. Each bus bridge in the server is required to support data bursting.

THE BUS BRIDGES

In the previous section, we discussed certain design techniques used to promote efficiency within the server's hierarchical bus structure. The section that follows describes the bus bridges in more detail, emphasizing a few interesting features.

The T2 Bridge Chip

The T2 bridge chip is a specially designed ASIC that provides bridge functionality between the server's multiprocessing system bus and the primary PCI bus. (See Figures 1 and 2.) The T2 ASIC is a 5.0-volt chip designed in complementary metal-oxide semiconductor (CMOS) technology. It is packaged in a 299-pin ceramic pin grid array (CPGA).

As stated earlier, the T2 bridge chip contains a 128-bit system bus interface running at 24 ns and a 32-bit PCI interface running at 30 ns. By using these two interfaces and data buffering, the T2 bridge chip translates bus protocols in both directions and moves data on both buses, thereby providing the logical system bus-to-PCI interface (bridge). In addition to the previously mentioned bridge features, the T2 bridge chip integrates system functions such as parity protection, error reporting, and CPU-to-PCI address and data mapping, which is discussed later in the section Connecting the Alpha CPU to the PCI and EISA Buses.

The T2 bridge chip contains a sophisticated DMA controller capable of servicing three separate PCI masters simultaneously. The DMA controller supports different-size data bursting (e.g., single, multiple, or continuous) and two kinds of DMA transfers, direct mapped and scatter/gather mapped. Both DMA mappings allow the T2 bridge chip to transfer large amounts of data between the PCI bus and the system bus, independent of the CPU.

Direct-mapped DMAs use the address generated by the PCI to access the system bus memory directly. Scatter/gather-mapped DMAs use the address generated by the PCI to access a table of page frame numbers (PFNs) in the system bus memory. By using the PFNs from the table, the T2 bridge chip generates a new address to access the data. To enhance the performance of scatter/gather-mapped DMAs, the T2 bridge chip contains a translation look-aside buffer (TLB) that contains eight of the most recently used PFNs from the table. By storing the PFNs in the TLB, the T2 bridge chip does not have to access the table in system bus memory every time it requires a new PFN. The TLB improves scatter/gather-mapped DMA performance and conserves bus bandwidth. Each entry in the TLB can be individually invalidated as required by software.

The T2 bridge chip also contains a single I/O data mover that enables a CPU on the system bus to initiate data transfers with a device on the PCI bus. The I/O data mover supports accesses to all the valid PCI address spaces, including PCI I/O space, PCI memory space, and PCI configuration space. The T2 bridge chip supports two I/O transaction types when accessing PCI memory space: sparse-type data transfers and dense-type data transfers. Sparse-type transfers are low-performance operations consisting of 8-, 16-, 24-, 32-, and 64-bit data transactions. Dense-type transfers are high-performance operations consisting of 32-bit through 32-byte data transactions. Dense-type transfers are especially useful when accessing I/O devices with large data buffers, such as video graphics adapter (VGA) controllers. A single PCI device mapped into PCI memory space can be accessed with either sparse-type operations, dense-type operations, or both.

In addition to accessing the PCI, a CPU can access various T2 bridge chip internal control/status registers (CSRs) for setup and status purposes. For maximum flexibility, all the T2 bridge chip's functions are CSR programmable, allowing for a variety of optional features. All CPU I/O transfers, other than those to T2 bridge chip CSRs, are forwarded to the PCI bus.

Intel PCI-to-EISA Bridge Chip Set

The Intel PCI-to-EISA bridge chip set provides the bridge between the PCI bus and the EISA bus.[3] It integrates many of the common I/O functions found in today's EISA-based PCs. The chip set incorporates the logic for a PCI interface running at a clock rate of 30 ns and an EISA interface running at a clock rate of 120 ns. The chip set contains a DMA controller that supports direct- and scatter/gather-mapped data transfers, with a sufficient amount of data buffering to isolate the PCI bus from the EISA bus. The chip set also includes PCI and EISA arbiters and various other support control logic that provide decode for peripheral devices such as the flash read-only memories (ROMs) containing the basic I/O system (BIOS) code, real-time clock, keyboard/mouse controller, floppy controller, two serial ports, one parallel port, and hard disk drive. In the AlphaServer 2100 system, the PCI-to-EISA bridge chip set resides on the standard I/O module, which is discussed later in this paper.

CONNECTING THE ALPHA CPU TO THE PCI AND EISA BUSES

In the next section, we discuss several interesting design challenges that we encountered as we attempted to connect PC-oriented bus structures to a high-powered multiprocessing Alpha chassis.

Address and Data Mapping

When a CPU initiates a data transfer to a device on the PCI bus, the T2 bridge chip must first determine the location (address) and amount of data (mask) information for the requested transaction and then generate the appropriate PCI bus cycle. This issue is not straightforward because the PCI and EISA buses both support data transfers down to the byte granularity, but the Alpha CPU and the system bus provide masking granularity only down to 32 bits of data.

To generate less than 32-bit addresses and byte-masked data transactions on the PCI bus, the T2 bridge chip needed to implement a special decoding scheme that converts an Alpha CPU-to-I/O transaction, as it appears on the system bus, to a correctly sized PCI transaction. Tables 1 and 2 give the low-order Alpha address bits and Alpha 32-bit mask fields and show how they are encoded to generate the appropriate PCI address and data masks. By using this encoding scheme, the Alpha CPU can perform read and write transactions to a PCI device mapped in either PCI I/O, PCI memory, or PCI configuration space with sparse-type transfers. (Sparse-type transfer sizes have 8-, 16-, 24-, 32-, or 64-bit data granularity.)

Table 1 CPU-to-PCI Read Size Encoding

Transaction Size	EV_Addr [6:5]	EV_Addr [4:3]	Instruc- tions	PCI Byte Enables (L)	PCI_AD [1:0]	Data Returned to Processor, EV_Data[127:0]
8 bits	00	00	LDL	1110	00	OW_0:[D7:D0]
	01	00	LDL	1101	01	OW_0:[D15:D8]
	10	00	LDL	1011	10	OW_0:[D23:D16]
	11	00	LDL	0111	11	OW_0:[D31:D24]
16 bits	00	01	LDL	1100	00	OW_0:[D79:D64]
	01	01	LDL	1001	01	OW_0:[D87:D72]
	10	01	LDL	0011	10	OW_0:[D95:D80]
24 bits	00	10	LDL	1000	00	OW_1:[D23:D0]
	01	10	LDL	0001	01	OW_1:[D31:D8]
32 bits	00	11	LDL	0000	00	OW_1:[D95:D64]
64 bits	11	11	LDQ	0000 0000	00	OW_1:[D95:D64] OW_1:[D127:D96]

Table 2 CPU-to-PCI Write Size Encoding

Transaction Size	EV_Addr [6:5]	EV_Addr [4:3]	EV_Mask [7:0] (H)	Instructions	PCI Byte Enables (L)	PCI_AD [1:0]	Data Returned to Processor, EV_Data[127:0]
8 bits	00	00	00000001	LDL	1110	00	OW_0:[D7:D0]
	01	00	00000001	LDL	1101	01	OW_0:[D15:D8]
	10	00	00000001	LDL	1011	10	OW_0:[D23:D16]
	11	00	00000001	LDL	0111	11	OW_0:[D31:D24]
16 bits	00	01	00000100	LDL	1100	00	OW_0:[D79:D64]
	01	01	00000100	LDL	1001	01	OW_0:[D87:D72]
	10	01	00000100	LDL	0011	10	OW_0:[D95:D80]
24 bits	00	10	00010000	LDL	1000	00	OW_1:[D23:D0]
	01	10	00010000	LDL	0001	01	OW_1:[D31:D8]
32 bits	00	11	01000000	LDL	0000	00	OW_1:[D95:D64]
64 bits	11	11	11000000	LDQ	0000 0000	00	OW_1:[D95:D64] OW_1:[D127:D96]

Another mapping problem exists when a PCI device wants to move a byte of data (or anything smaller than 32 bytes of data) into the system bus memory. Neither the system bus nor its memory supports byte granularity data transfers. Therefore, the T2 bridge chip must perform a read-modify-write operation to move less than 32 bytes of data into the system bus memory. During the read-modify-write operation, the T2 bridge chip first reads a full 32 bytes of data from memory at the address range specified by the PCI device.[2] It then merges the old data (read data) with the new data (PCI write data) and writes the full 32 bytes back into memory.

ISA Fixed-address Mapping

We encountered a third interesting mapping problem when we decided to support certain ISA devices with fixed I/O addresses in the AlphaServer 2100 system. These ISA devices (e.g., ISA local area network (LAN) card or an ISA frame buffer) have fixed (hardwired) memory-mapped I/O addresses in the 1-MB to 16-MB address range.

The ISA devices being discussed were designed for use in the first PCs, which contained less than 1 MB of main memory. In these PCs, the I/O devices had fixed access addresses above main memory in the 1-MB to 16-MB address range. Today's PCs have significantly more physical memory and use the 1-MB to 16-MB region as a part of main memory. Unfortunately, these ISA devices were never redesigned to accommodate this change. Therefore, to support these ISA options, the PC designers created I/O access gaps in main memory in the 1-MB to 16-MB address range. With this technology, an access by a CPU in that address range is automatically forwarded to the ISA device.

To remain compatible with the ISA community, the T2 bridge chip also had to allow for a gap in main memory at the 1-MB to 16-MB address range so that these addresses could be forwarded to the appropriate ISA device.

BIOS CACHING COMPATIBILITY

Today's Microsoft-compatible PCs provide another performance-enhancing mechanism. We decided to implement this function inside the T2 bridge chip as well.

During system initialization, MS-DOS-based PCs read several BIOS ROMs from their I/O space. Once the ROMs are read, their contents are placed in fixed locations in main memory in the 512-kilobyte (KB) to 1-MB address range. The software then has the ability to mark certain addresses within this range as read cacheable, write cacheable, read noncacheable, or write noncacheable. The basic intention is to mark frequently accessed sections of code as read cacheable but write noncacheable. In this way, read accesses "hit" in main memory (or cache), and writes update the ROMs directly.

INTERRUPT MECHANISM

No computer system would be complete without providing a mechanism for an I/O device to send interrupts to a CPU. The I/O interrupt scheme on the AlphaServer 2100 system combines familiar technology with custom support logic to provide a new mechanism.

Electrical and architectural restrictions prohibited the interrupt control logic from being directly accessed by either the system bus or the PCI bus. As a result, the interrupt control logic is physically located on a utility bus called the XBUS. The XBUS is an 8-bit slave ISA bus placed nearby the PCI-to-EISA bridge chips.

The base technology of the I/O interrupt logic is a cascaded sequence of Intel 8259 interrupt controllers. The 8259 chip was chosen because it is a standard, accepted, and well-known controller used by the PC industry today. The use of the 8259 interrupt controller translated to low design risk as well. Although the 8259 interrupt controller is not new, its integration into a high-performance multiprocessing server, without incurring undue performance degradation, required some novel thinking.

The integration of the 8259 interrupt controller into the AlphaServer 2100 system presented two considerable problems. First, the designers had to satisfy the 8259 interface requirements in a way that would have a minimal impact on the performance of the interrupt-servicing CPU. The 8259 requires two consecutive special-acknowledge cycles before it will present the interrupt vector. To resolve this problem, we designed a set of handshaking IACK programmable array logic (PAL) devices. These PALs enhance the functions of the 8259 controllers as XBUS slaves. The interrupt-servicing CPU performs only a single read to a designated address that is decoded to the XBUS. The IACK-control PALs decode this read and then generate the special, double-acknowledge cycles required to access the vector. The PAL logic also deasserts CHRDY, a ready signal to the ISA bus, so that the cycle has ample time to proceed without causing a conformance error for a standard ISA slave cycle. When the double acknowledge is complete and the vector is guaranteed to be driven on the bus, the PALs assert the CHRDY ready signal.

The second problem involved the location of the interrupt controller. As mentioned earlier, because of electrical and architectural restrictions, the interrupt controller was located on the XBUS near the PCI-to-EISA bridge chips. With the interrupt controller located on the XBUS, an interrupt-servicing CPU is required to perform a vector read that spans two I/O bus structures. For this reason and its potential effect on system performance, vector reads had to be kept to a minimum, which is not easy in a system that allows more than one CPU to service a pending interrupt request.

Since the AlphaServer 2100 system can have as many as four CPUs, all four CPUs can attempt to service the same pending interrupt request at the same time. Without special provisions, each CPU would perform a vector read of the interrupt controller only to find that the interrupt has already been serviced by another CPU. Requiring each CPU to perform a vector read of the interrupt controller on the XBUS wastes system resources, especially when each vector read spans two bus structures. Of course, this problem could be resolved by assigning only one CPU to service pending interrupts, but this would negate the advantage of having multiple CPUs in a system. To solve this problem, the T2 bridge chip on the system bus implements special "passive-release" logic that informs a CPU at the earliest possible time that the pending interrupt is being serviced by another CPU. This allows the "released" CPU to resume other, more important tasks.

The term passive release typically refers to a vector code given to an interrupt-servicing CPU during a vector read operation. The passive-release code informs the CPU that no more interrupts are pending. The special passive-release logic allows the T2 bridge chip to return the passive-release code to a servicing CPU on behalf of the interrupt controller. The T2 bridge chip performs this function to save time and bus bandwidth.

After the designers implemented all the features described above, they needed to address the problem of how to deal with all the slow, highly volatile, "off-the-shelf" parts. To integrate these components into the I/O subsystem, they invented the standard I/O module.

THE STANDARD I/O MODULE

As part of the development effort of the I/O subsystem, the engineering team faced the challenge of integrating several inexpensive, low-performance, off-the-shelf, PC-oriented I/O functions (e.g., TOY clock, keyboard, mouse, speaker) into a high-performance Alpha multiprocessing system, without affecting the higher performing architectural resources. The multilevel I/O bus structure served to alleviate the performance issues, but the development of a PC-style I/O subsystem with off-the-shelf components involved inherent risk and challenge.

To reduce the risks inherent with using new and unfamiliar devices, such as the PCI-to-EISA bridge chip set, we chose to build an I/O module (called the standard I/O module) that plugs into the AlphaServer 2100 system backplane and contains the PCI-to-EISA bridge, associated control logic, controllers for mouse, keyboard, printer, and floppy drive as well as the integral Ethernet and SCSI controllers. Without this plug-in module, fixing any problems with the PCI-to-EISA bridge chip set or any of the supporting logic would have required a backplane upgrade, which is a costly and time-consuming effort.

The standard I/O module is relatively small, inexpensive both to manufacture and to modify, and easily accessible as a field replaceable unit (FRU). As shown in Figure 3, the standard I/O module contains the following logic:

o PCI-to-Ethernet controller chip

o PCI-to-SCSI controller chip

o PCI-to-EISA bridge chips

o Real-time clock speaker control

o 8-KB, nonvolatile, EISA-configuration, random-access memory (RAM)

- 1-MB BIOS flash ROM
- Keyboard and mouse control
- Parallel port
- FDC floppy controller
- Two serial ports
- I**2C support: controller, expander, and ROM
- Intel 8259 interrupt controllers
- Ethernet station address ROM
- Reset and sysevent logic
- Fan speed monitor
- Remote fault management connector
- External PCI subarbiter
- 3.3-volt and --5.0-volt generation

[Figure 3 (The Standard I/O Module) is not available in ASCII format.]

For the most part, all these functions were generated by using integrated, off-the-shelf components at commodity pricing. Solutions known to work on other products were used as often as possible. The flash memory resides on the EISA memory bus and is controlled by the PCI-to-EISA bridge chip. A simple multiplexing scheme with minimal hardware enabled the server to address more locations than the bridge chip allowed, as much as a full 1 MB of BIOS ROM. The National PC87312, which provides the serial and parallel port control logic, and the floppy disk controller reside directly on the ISA bus. The rest of the devices are located on the XBUS (an 8-bit buffered slave ISA bus), with control managed by the PCI-to-EISA bridge chips.

In addition, the common PC functions are located at typical PC addresses to ease their integration and access by software. As expected, hardware changes were required to the standard I/O module during its hardware development cycle. However, the standard I/O module, which takes only minutes to replace, provided an easy and efficient method of integrating hardware changes into the AlphaServer 2100 system. We expect the usefulness of the standard I/O module to continue and hope that it will provide an easy and inexpensive repair process.

SUMMARY

The I/O subsystem on the AlphaServer 2100 system contains a two-level hierarchical bus structure consisting of a high-performance PCI bus connected to a secondary EISA bus. The PCI bus is connected to the AlphaServer 2100 system's multiprocessing system bus through the T2 bridge chip. The secondary I/O bus is connected to the PCI bus through a standard bridge chip set. The I/O subsystem demonstrated sufficient flexibility to become the I/O interface for the small pedestal AlphaServer 2000 and the rackmountable version of the AlphaServer 2100 products.

ACKNOWLEDGMENTS

The AlphaServer 2100 I/O would not be what it is today, without the dedicated, focused efforts of several people. Although not complete, the following list gives credit to those who were the most deeply involved. Thanks to Fidelma Hayes for leading the Sable effort; to Vicky Triolo for the Sable mother board and her support of the T2 bridge chip effort; to Rachael Berman for her unflagging support of the standard I/O module; to Lee Ridlon for his much needed early conceptual contributions; to Stephen Shirron for driving Sable software I/O issues; to John Bridge for cleaning up the second-pass T2; and to Tom Hunt and Paul Rotker for their contributions to the first-pass T2.

REFERENCES

1. F. Hayes, "Design of the AlphaServer Multiprocessor Server Systems," Digital Technical Journal, vol. 6, no. 3 (Summer 1994): 8-19.

2. PCI Local Bus Specification, Revision 2.0 (Hillsboro, OR: PCI Special Interest Group, Order No. 281446-001, April 1993).

3. 82420/82430 PCIset ISA and EISA Bridges (Santa Clara, CA: Intel Corporation, 1993).

4. E. Solari, ISA and EISA, Theory and Operation (San Diego, CA: Annabooks, 1992).

BIOGRAPHY

Andrew P. Russo Andy Russo is a principal hardware engineer in the Alpha Server Group. Since joining Digital in 1983, Andy has been a project leader for several internal organizations, including the Mid-range I/O Options Group, the Fault Tolerant Group, and the Alpha Server Group. While at Digital, Andy has contributed to the architecture and design of high-performance ASICs and modules to provide a variety of end-product requirements. Andy holds several patents and has authored two papers. He received a B.S. in computer engineering from Boston University.

TRADEMARKS

Alpha and AlphaServer are trademarks of Digital Equipment Corporation.

Intel is a trademark of Intel Corporation.

Microsoft and MS-DOS are registered trademarks of Microsoft Corporation.

PAL is a registered trademark of Advanced Micro Devices, Inc.